Always-On Personal Intelligence:
A Roadmap Towards Continuing, Self-Evolving Loops

Shuai Liu[1,†], **Hongming Guo**[5], **Weiyang Jin**[1], **Kairui Hu**[1], **Shulin Tian**[1], **Yuhao Dong**[1], **Jinming Wu**[1], **Kaichen Zhang**[1], **Yujin Tang**[7], **Xinyang Han**[2], **Wenhao Chai**[3], **Yuqian Fu**[6], **Ce Zhang**[8], **Yuanhan Zhang**[1,‡], **Lingdong Kong**[5,‡], **Li Siyao**[1,‡], **Jingkang Yang**[1,‡], **Gedas Bertasius**[8,◇], **Manling Li**[9,◇], **Jiajun Wu**[4,◇], **Mengdi Wang**[3,◇], **Dawn Song**[2,◇], **Ziwei Liu**[1,✉]

[1]S-Lab, NTU, [2]UC Berkeley, [3]Princeton, [4]Stanford, [5]NUS, [6]KAUST, [7]Dartmouth, [8]UNC Chapel Hill, [9]Northwestern

[†]Project lead [‡]Equal mentorship [◇]Advisory committee [✉]Corresponding author: ziwei.liu@ntu.edu.sg

Project Page: https://always-on-intelligence.github.io

GitHub: https://github.com/Always-On-Intelligence/awesome-always-on-personal-intelligence

Abstract

Always-on personal intelligence demands more than episodic, task-driven agents. To remain always-on, it requires a *continuing loop* that sustains assistance between requests, retains personal context, and carries unfinished commitments forward. To grow more personal over time, this loop can also become *self-evolving*, revising how it works based on past experience. Yet sustaining and improving this loop as a whole remains an open challenge. We therefore organize the assistant into three functional modules, *Sensing*, *Memory*, and *Action*, connected by six directed channels that pass information between them. These modules operate under real-world limits, from battery life to privacy and user permissions, and run on models and the scaffolding around them. We review each module's capabilities, policies, and evaluation methods, and then trace cross-module dependencies across integrated systems to identify bottlenecks. To assess the loop completeness of existing systems, we introduce the *Harness for Always-On Loop Oversight* (HALO), combining literature and code reviews with executable tests. Our synthesis reveals the thinnest support for two channels: Memory guiding Sensing, and Action outcomes returning to Memory. These gaps motivate two research priorities: keeping sensing open to unexpected change for reliable operation, and attributing outcomes to earlier decisions for self-evolution. By identifying missing interfaces and evidence requirements, this roadmap offers the research community an actionable blueprint for building future always-on personal intelligence systems.

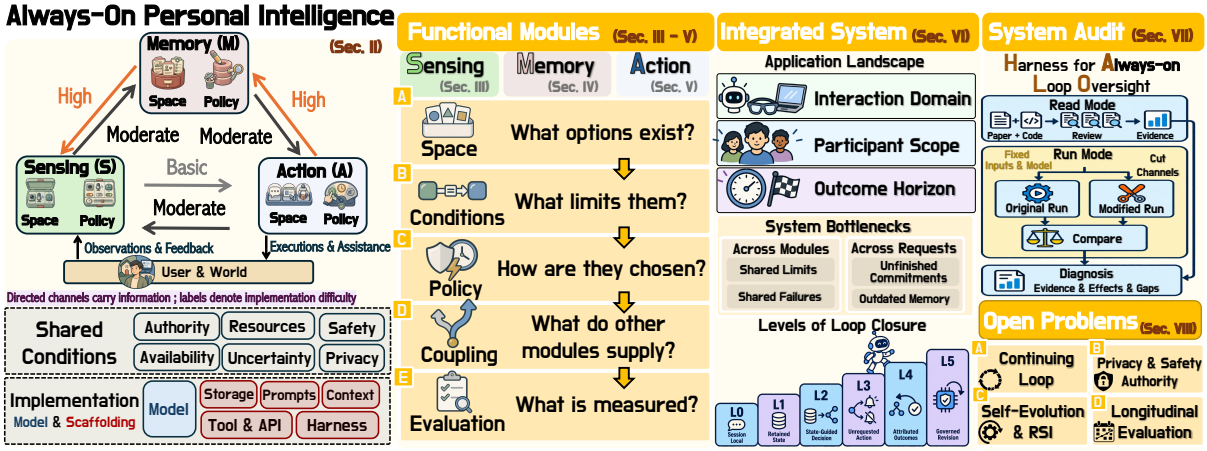


Figure 1 Overview of the roadmap. This roadmap is organized into five parts. We first define always-on personal intelligence as a continuing loop that connects Sensing, Memory, and Action through six directed channels (§2). We then review each module by its space, conditions, policy, coupling, and evaluation (§§3–5). Building on these reviews, we examine integrated systems, their bottlenecks, and their levels of loop closure (§6), and audit existing systems with HALO in read and run modes (§7). Finally, we discuss the open problems in always-on personal intelligence (§8).

Contents

1 Introduction . . . 3
- 1.1 Positioning and Related Surveys . . . 4
- 1.2 Contributions . . . 5
- 1.3 Main Takeaways at a Glance . . . 6

2 Definitions and Foundations . . . 8
- 2.1 Continuing and Self-Evolving Loops . . . 9
- 2.2 Functional Modules: Spaces and Local Policies . . . 11
- 2.3 Six Coupling Channels . . . 12
- 2.4 Shared Conditions and Implementation . . . 12
- 2.5 Application Scope and Adjacent Concepts . . . 14
- 2.6 Mapping the Literature . . . 15

3 Sensing . . . 17
- 3.1 Sensing Space . . . 17
- 3.2 Shared Conditions for Sensing . . . 21
- 3.3 Sensing Policy . . . 22
- 3.4 Sensing Coupling Across the Loop . . . 26
- 3.5 Evaluation . . . 29

4 Memory . . . 33
- 4.1 Memory Space . . . 33
- 4.2 Shared Conditions for Memory . . . 36
- 4.3 Memory Policy . . . 37
- 4.4 Memory Coupling Across the Loop . . . 41
- 4.5 Evaluation . . . 43

5 Action . . . 46
- 5.1 Action Space . . . 46
- 5.2 Shared Conditions for Action . . . 49
- 5.3 Action Policy . . . 50
- 5.4 Action Coupling Across the Loop . . . 53
- 5.5 Evaluation . . . 55

6 Applications, System Bottlenecks, and Loop Closure . . . 58
- 6.1 Applications and Bottlenecks . . . 58
- 6.2 Shared Conditions and Implementation Across Modules . . . 62
- 6.3 Levels of Loop Closure . . . 65
- 6.4 Self-Evolution in Deployed Loops . . . 70

7 HALO: Harness for Always-On Loop Oversight . . . 72
- 7.1 Read Mode: Channel Coverage and Evidence for Self-Evolution . . . 72
- 7.2 Run Mode: Testing Channels and the Chain of Evidence . . . 74

8 Trends and Open Problems . . . 76
- 8.1 Sustaining the Continuing Loop . . . 76
- 8.2 Authority, Privacy, and Safety . . . 81
- 8.3 Self-Evolution and Recursive Revision . . . 84
- 8.4 Longitudinal Evaluation and Real Deployment . . . 87

9 Conclusion . . . 88

1 Introduction

> *The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.*
>
> Weiser [1]

Personal intelligence is moving from answering isolated requests to providing sustained assistance. Research toward this goal spans physical assistants in everyday tasks [2, 3, 4, 5, 6], wearable assistants with live egocentric context [7, 8, 9, 10], and digital assistants across applications [11, 12, 13, 14, 15, 16, 17]. Across these devices and settings, useful assistance connects current situations with earlier interactions [18, 19, 20], continues after an interaction ends, and adapts to the individual user over time.

What must continue when an interaction ends? Suppose a user asks the assistant to remind them to send a report after a meeting. Although the conversation has ended, the assistant must follow up in the background, detect when the meeting ends, and deliver the reminder at an appropriate time. Sustaining this assistance across interactions requires a *continuing loop*. Within this loop, the assistant remains operational after the user stops interacting, senses changes through events or timed checks, and acts when appropriate. The assistant also preserves relevant personal state across interactions, including personal context, task progress, and commitments, defined as pending tasks the assistant has agreed to carry out for the user. Together, these two properties, *operational continuity* and *state durability*, form the foundation of always-on personal intelligence.

Building on this continuing loop, the long-term goal is a *self-evolving loop*, in which the assistant revises how it works based on operating experience to serve the user better. In the always-on setting, this experience comes naturally from repeated daily use by the same user, but the outcomes of the assistant's decisions are often delayed, sparse, or even implicit. Returning to the report reminder, suppose that the user usually dismisses such reminders right after meetings and sends the report only once they are back at their workstation. The assistant attributes each dismissal to the decision to send the reminder as soon as the meeting ends, and to the timing rule behind that decision. Based on this attribution, the assistant can revise the timing rule to wait until the user is back at their workstation, instead of simply sending fewer reminders. Throughout this process, the user never has to tell the assistant when to send reminders, yet the assistant becomes more effective the more it is used. This example points toward the end state we envision: the user steers the loop instead of running it, setting only its direction and boundaries, while the assistant decides when to act, revises how it works, and asks for confirmation whenever an action needs authorization from the user.

Progress toward these goals spans several research areas. Memory-management research examines how personal context is retained and updated [18, 19, 20], proactive assistance studies how observations inform response decisions [12, 21, 22], and multi-session evaluations investigate how retained experience supports later actions [23, 24]. These studies analyze individual methods and selected interactions, but the open challenge is how these advances work together to support a continuing loop and enable reliable self-evolution. Understanding why an assistant fails and what should change requires tracing dependencies across the loop.

As shown in Figure 1, we organize this roadmap around three functional modules, *Sensing*, *Memory*, and *Action*, and the *coupling channels* between them. Within each module, we distinguish its *space*, what the module can do and represent, from the *local policy* that chooses among these options. Across modules, six possible coupling channels each carry information from one module to another for the receiving module to use. We define shared conditions as the limits set by the user and the world, including privacy, user authority, safety, and resource budgets. Within these limits, every module and channel runs on an implementation, which keeps and passes the information, makes each module's decisions, and carries them out. We define the implementation as the models and the scaffolding around them, such as prompts, tools, memory stores, and the harness that runs the loop. In the reminder example, the user's calendar permission and the phone's battery are shared conditions. The language model, the calendar tool, and the store that keeps the pending reminder form the implementation. Using this framework, §6 analyzes existing systems as a whole, tracing dependencies across modules, and §7 introduces the *Harness for Always-On Loop Oversight* (HALO) to assess their loop completeness. Finally, §8 groups the open problems that remain into four parts: sustaining

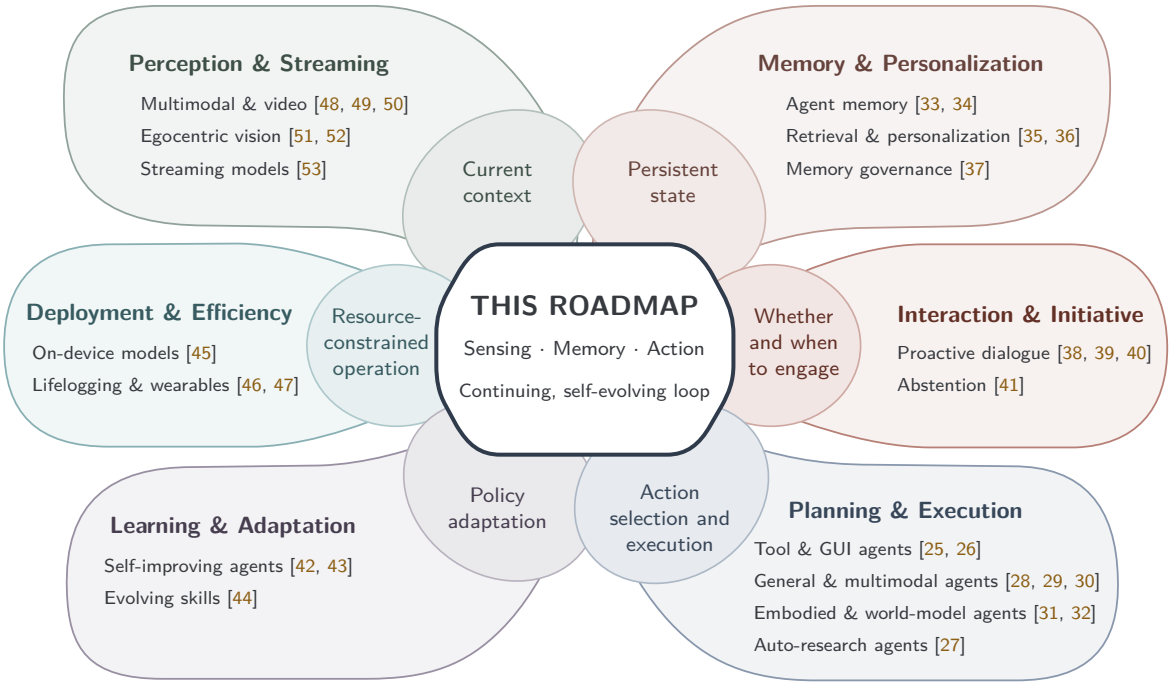


Figure 2 Topical overlap with capability-focused surveys. Outer groups show survey themes, and inner labels mark topics shared with this roadmap. Table 1 compares concurrent system-level surveys.

the continuing loop; authority, privacy, and safety; self-evolution and recursive revision; and longitudinal evaluation in real deployment.

Our synthesis reveals the thinnest support for two channels: Memory guiding Sensing (M$\rightarrow$S), and action outcomes returning to Memory (A$\rightarrow$M). Both are hard because their errors can hide the evidence that would correct them. An outdated memory can steer the assistant's next observation away from the change that would correct it, so wrong suggestions or actions follow. Feedback tied to the wrong decision can likewise change the wrong rule. These gaps motivate two research priorities. First, keeping sensing open to unexpected change makes the loop reliable even when retained state is outdated. Second, attributing each outcome to the decision and rule behind it lets the loop evolve correctly, even when outcomes arrive late, sparsely, or only implicitly.

1.1 Positioning and Related Surveys

This roadmap examines always-on personal intelligence systems and applications across physical, wearable, and digital settings, where personal assistance centers on a particular user's state, goals, and preferences. We ask how the Sensing, Memory, and Action modules work together as an assistant continues to assist the same user: what information each needs from the others, and how missing or unreliable information affects later assistance.

Capability-focused surveys. Figure 2 shows the topics this roadmap shares with capability-focused surveys. These surveys cover perception across modalities and deployment settings [48, 49, 51, 52, 47, 53, 46, 45], and how information is retained, used, and governed [33, 37, 35, 36]. Other surveys examine proactive assistance [38, 39, 40, 54] and agent decision and execution methods [28, 29, 30, 25, 26, 31, 32], while surveys of self-improving agents address changes to models and control logic [42, 55, 43, 44, 56]. Each of them studies one capability in depth, and several also discuss integration. Yet an improvement in one capability does not by itself yield a better assistant. For example, retaining a user's preferences accurately does not make a reminder

timely: Action must also combine those preferences with fresh evidence from Sensing about the user's current situation. Our framework makes such dependencies explicit, so it can show when an improvement within one module helps the assistant and when another module limits its benefit. A detailed taxonomy of any single capability, such as visual recognition or robot control, is not included in the scope of this roadmap.

Concurrent system-level surveys. Table 1 compares this roadmap to concurrent surveys, which organize the field from different perspectives. *ComBodied Agents* [57] proposes a human-centric paradigm of agents that act on a person's own state, from physiology to cognition, and are judged by the person's long-term benefit. *AI Smart Glasses* [58] and *From Seeing to Acting* [59] center on the device, from its hardware and capabilities to interaction and applications. *Long-Horizon Agents* [60] focuses on tasks with many dependent steps and organizes the field by implementation, from the training of the model to the scaffolding around the model. *Proactive AI* [61] organizes proactiveness, acting without a new request within a delegated task, into six capacities across 13 domains. Proactiveness resembles operational continuity, yet an assistant that helps within one conversation and keeps nothing afterward also has it, so proactiveness alone does not make assistance always-on.

The closest is *Always-On Agents* [62], which follows what an agent keeps across sessions, like preferences and unfinished tasks, over its lifecycle from creation to deletion. The survey covers in depth how retained state is governed, including deletion propagation, rollback, and memory shared across agents, but this lifecycle view does not show how Sensing, Memory, and Action depend on one another. Because it follows stored items, it does not cover the decisions made between requests, such as whether to act now or stay silent, so an agent that acts only when the user opens a session still counts as always-on. This roadmap instead centers on the loop that connects the three modules and on the information flow between them. Following these flows shows where an error starts: a reminder that arrives late, for example, can start with Sensing missing the end of the meeting, not with Action. The same flows show what the assistant does between requests, and when one of them fails, they point to the mechanism a self-evolving assistant should revise, here the rule that decides when Sensing checks again.

Table 1 Concurrent surveys: scope and organizing frameworks. Icons indicate deployment coverage: physical, wearable, and digital.

Survey	Domain	Research focus	Organizing framework	URL
AI Smart Glasses [58]		**Wearable intelligence**	Co-design of hardware, intelligence, interaction, and applications	/GitHub
ComBodied Agents [57]		**Human-centric longitudinal support**	Closed loop of human-centric perception, memory, world models, and intervention	/–
From Seeing to Acting [59]		**Capabilities of first-person intelligence**	Capability levels from perception to action, and hardware conditions	/GitHub
Long-Horizon Agents [60]		**Tasks with tightly coupled steps**	Coupled model–harness co-evolution across three task levels	/GitHub
Proactive AI [61]		**Self-initiated action**	Six capacities, from awareness to adaptation	/–
Always-On Agents [62]		**State kept across sessions**	Lifecycle of an agent's persistent state	/–
This roadmap		**Always-on personal assistance**	Sensing–Memory–Action modeling with information flows	

1.2 Contributions

This roadmap makes three contributions:

- **A unified framework for always-on personal intelligence.** We define the continuing loop as the foundation of always-on personal intelligence and specify the requirements for its self-evolving extension. We then decompose the loop into three functional modules, Sensing, Memory, and Action, and identify the six possible directed coupling channels between them. For each module, we analyze its space and local policy. We also examine the shared conditions that apply to every module and channel, and the implementation that passes information and carries out the assistant's decisions. This decomposition helps

locate failures of the continuing loop in specific modules or channels and identify what a self-evolving assistant should revise.

- **A system-level synthesis and roadmap.** We map existing always-on and related work onto our framework to provide an integrated view of current systems. To compare their capabilities and guide further development, we introduce an L0–L5 classification of loop closure. We also analyze the implementation challenges of all six coupling channels and identify $\mathrm{M} \to \mathrm{S}$ and $\mathrm{A} \to \mathrm{M}$ as the most challenging, making them priorities for reliable operation and self-evolution.
- **HALO for reproducible system analysis.** We introduce HALO with read and run modes to analyze individual systems through their coupling channels and the chain of evidence for self-evolution. In read mode, a panel of three independent oversight agents reads each system's paper or blog and any released code. A majority vote then decides which channels each system reports and which steps of that chain its sources state. In run mode, HALO runs executable systems in controlled tests, blocks channels or evidence steps singly or together, and checks how their behavior changes. HALO thus gives researchers and teams deploying always-on assistants an executable prototype for analyzing and comparing their systems.

1.3 Main Takeaways at a Glance

Definitions and framework (§2)

- **Always-on assistance rests on a continuing loop.** The assistant needs two properties. *Operational continuity* lets it decide by itself whether and when to act, without waiting for a new request. *State durability* lets it keep usable personal state, such as preferences and pending tasks, across interactions. A loop of sensing, remembering, and acting that keeps running between requests provides both, while an assistant that remembers but waits for every request has only the second.
  §2.1 *Continuing and Self-Evolving Loops*
- **Self-evolution extends the loop, and it needs evidence.** To serve the user better, the assistant learns from its own trace and the user's feedback to change how it works, such as a rule, a prompt, or its model weights, while it keeps running. Self-evolution is optional for an always-on assistant, but it is the path to a more personal system. A claim of self-evolution needs evidence that the assistant recorded what it did, observed the outcome, linked it to the decision behind it, changed the responsible rule, and used that rule later, all within the user's limits. §2.1.2 *Self-Evolving Extension*
- **Three modules connected by six channels.** Sensing observes, Memory retains what later decisions need, and Action decides whether, when, and how to help. Six coupling channels are the information flows between them, and each module separates what it can do from how it chooses. Shared limits, such as privacy, permissions, and budgets, constrain this information, and the same models and software process it. Modeling the flows shows where a failure starts, which can be in another module, and which part a self-evolving assistant should revise. §2.2 *Functional Modules: Spaces and Local Policies*

Sensing, Memory, Action, and how they connect (§§3–5)

- **Sensing: misses go unnoticed.** Missing an event can do more harm than misreading it: the assistant does not know what it missed, so nothing prompts it to correct its state or its sensing policy. Power limits keep always-on capture at low frame rates, so brief actions can fall between frames, yet most evaluations test recognition or timing on prerecorded streams and do not count what was missed. §3.5 *Evaluation*
- **Memory: errors get reused and can reinforce themselves.** A wrong record can be reused before the user notices it, so a correction has to reach every summary and commitment built on it. Faulty memory can also bias how later feedback is read, so each revision keeps or even amplifies the error. We call this a memory trap, and no reviewed always-on system yet shows lasting protection against it. §4.5, §4.4.3

- **Action: outcomes come late, and help can become a burden.** A message that an action was sent does not show that it worked, and the real outcome often arrives late and rarely. The assistant also has to keep commitments, the tasks it has promised to do, after the session or device that created them is gone, yet a reminder that helps today can become a nuisance when it is repeated. §5.5 *Evaluation*

- **Memory rarely guides observation, and outcomes rarely return to Memory.** Assistants rarely use memory to decide what to observe next (M→S), and when they do, an outdated memory can keep them from the evidence that would correct it. Many works record their actions, but fewer keep the outcomes (A→M), which arrive late or partial and must first be matched to their decision. Keeping sensing open to unexpected change and linking each outcome to the decision and rule behind it are therefore the two research priorities. §3.4.2, §4.4.3

What current systems show (§6)

- **Bottlenecks and shared limits span modules.** On wearables, what is captured now limits what can be recalled later, and in digital assistance, background checks compete with responses for the same computation. In physical assistance, a wrong action costs more, and judging whether help worked needs Memory to keep the user's response. A shared limit must also hold in every module at once: in a confidential meeting, the assistant may listen but must not store or repeat what it hears, yet no reported system controls all three modules. §6.1, §6.2.1

- **Most assistants stop short of the full loop.** Of the 119 systems we place, 52 are not yet always-on: they keep nothing across sessions, or they keep state but wait for requests or act only on a fixed schedule. Of the 67 that decide by themselves when to act, about half link an outcome to the decision behind it, mostly through explicit responses such as ratings or dismissals. Only five research works keep every change to how they work within the user's limits, each only under a stated condition. §6.3 *Levels of Loop Closure*

- **Changing itself is harder inside a running assistant.** A change cannot be tried cheaply, tasks promised under the old rule are still pending, and a revised rule also changes what the assistant observes next. No reported system shows in full how a changed rule performs later, and none changes the rules that decide when it wakes. §6.4 *Self-Evolution in Deployed Loops*

How HALO tests systems (§7)

- **HALO reads and runs.** In read mode, three language-model reviewers read the paper or blog and any released code of each of 40 works and vote on which channels and evidence steps it reports. No work that reports all channels also changes how it works, so the two must be judged separately. In run mode, HALO reruns executable systems with a channel or step removed. In a smart-glasses assistant we tested, the user profile reached the suggestions by two paths, so channels must be cut alone and together. §7 *HALO: Harness for Always-On Loop Oversight*

What remains open, and the goal (§8)

- **The goal: the user steers, and the assistant runs the loop.** The user sets only the loop's direction and limits, while the assistant decides when to act, revises how it works, and asks for confirmation whenever an action needs the user's permission. Reaching this goal requires solving four groups of open problems: keeping the loop running between requests, protecting authority, privacy, and safety while the user is away, checking self-evolution and keeping it within the user's limits, and evaluating assistants over months of real use. §8 *Trends and Open Problems*

2 Definitions and Foundations

In this section, we introduce the framework of always-on personal intelligence, as shown in Figure 3. We first define the continuing loop and its self-evolving extension. We then decompose the assistant into Sensing, Memory, and Action, describe each module through its space and local policy, and identify six possible directed coupling channels. The environment, which includes the user and the world, lies outside these modules: the Sensing module takes in observations and feedback from the environment, and the Action module returns executions and assistance to it. In addition, every module and channel is subject to shared conditions, limits such as privacy, user permissions, and battery life. Next, we describe these conditions and the implementation that passes information and carries out the assistant's decisions.

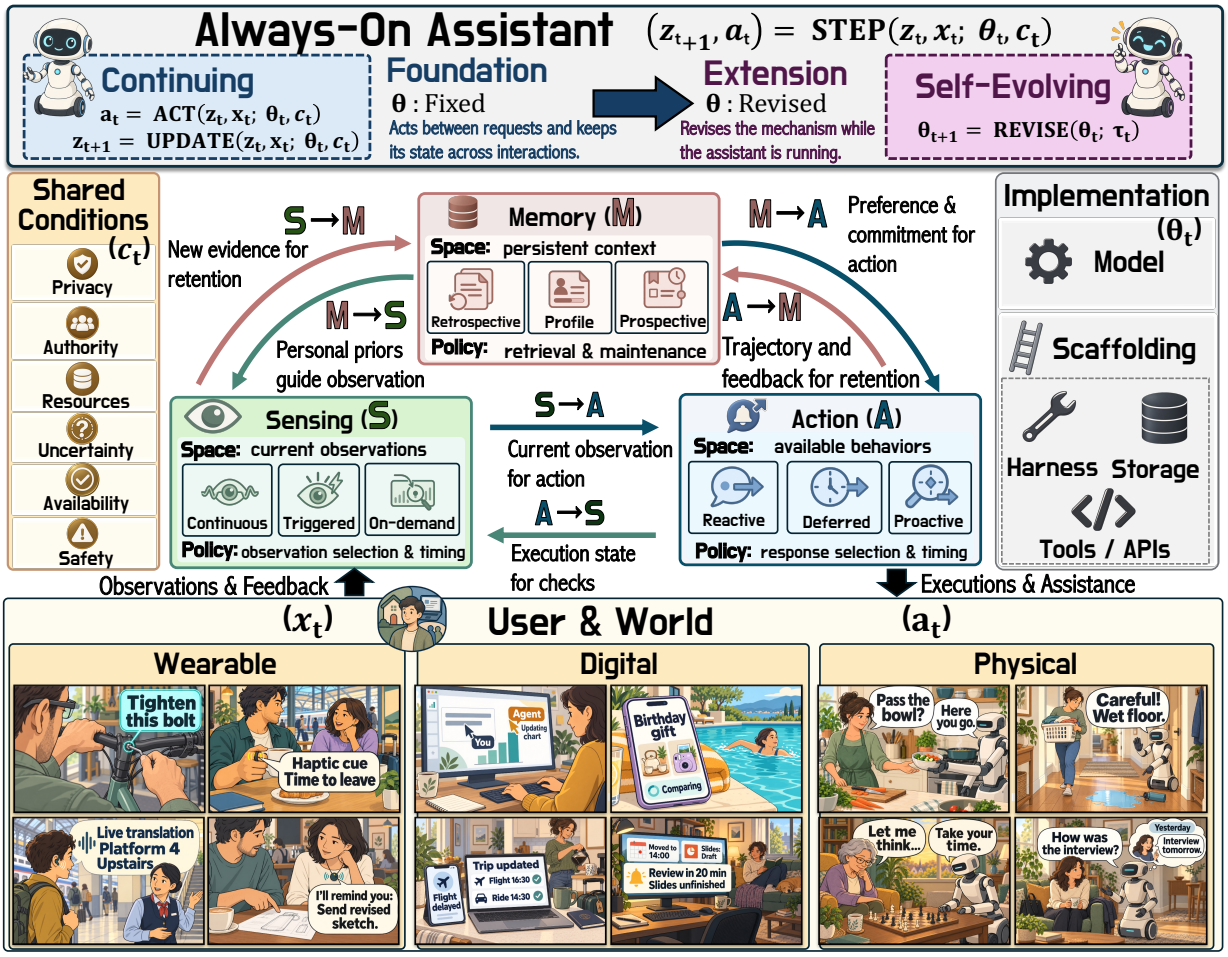


Figure 3 A framework for always-on personal intelligence. *Top:* the continuing loop is defined by *operational continuity* and *state durability*. Its self-evolving extension adds durable mechanism revision from operating experience. *Middle:* the assistant is decomposed into functional modules linked by directed information flows. Shared conditions constrain their operation, while models and scaffolding provide the implementation. *Bottom:* wearable, digital, and physical cases illustrate how the framework abstracts common functions and dependencies from different always-on applications. The user and the world are modeled separately from the functional modules: they provide observations and feedback, receive executions and assistance, and set the shared conditions.

2.1 Continuing and Self-Evolving Loops

Definition: Always-on personal intelligence

Always-on personal intelligence is personal assistance that maintains two core properties for the same user: *operational continuity* and *state durability*. *Operational continuity* is the ability to keep operating between explicit requests and to decide when to act or wait. *State durability* is the ability to preserve user-specific state for later use and to carry unresolved commitments across interaction boundaries. We define the *continuing loop* as the foundation of always-on personal intelligence. The loop provides both properties through the recurring operation of Sensing, Memory, and Action.

Symbol	Meaning
Continuing loop	
$t \in \mathcal{T}$	Step of the assistant: one runtime event or decision, and $\mathcal{T}$ is the set of all steps.
$\mathcal{R}$	Steps that carry out an explicit request as asked.
$\mathcal{B}$	Steps ending an interaction.
x_t	Input or event at step t.
a_t	Action chosen at step t, including waiting.
z_t	Running state: all information the assistant carries into step t, except the mechanism θ_t.
z_0	Initial running state.
τ_t	Trajectory: the assistant's inputs and actions up to step t, with feedback on them.
STEP	One assistant step: ACT and UPDATE together.
ACT	Chooses the assistant's current action.
UPDATE	Chooses the information kept for later steps.
REVISE	Revises the mechanism θ_t, changing how the assistant works.
Modules and channels	
π_i	Local policy of module $i \in \{S, M, A\}$.
$I_{i,t}$	Decision context of module i: external input, local state, and input from other modules.
$y_t^{i \to j}$	Information delivered from module i to module j.
Conditions and implementation	
c_t	Shared conditions: privacy, authority, safety, resources, availability, and uncertainty.
$\theta_t, \omega_t, \Sigma_t$	Mechanism θ_t implementing the local policies: model parameters ω_t and scaffolding Σ_t.

2.1.1 Continuing Loop

The continuing loop repeats Sensing, Memory, and Action both when the user makes a request and between requests. Each runtime event or decision is a step, indexed by t. At each step, the assistant as a whole produces two outputs: an action a_t, which can be to wait, and the next state z_{t+1}. We write this operation of an always-on assistant as $(z_{t+1}, a_t) = \text{STEP}(z_t, x_t; \theta_t, c_t)$. Here x_t is the input or event that the assistant receives at step t, and z_t is its running state, the information kept from earlier steps. The *mechanism* θ_t is the implementation written as a parameter of STEP, and it processes x_t and z_t to produce both outputs. It consists of the model parameters ω_t, the model's learned weights, and the scaffolding Σ_t, the software around the model. The mechanism also implements each module's local policy, written $\pi_i^{\theta_t}$. The shared conditions c_t are the limits that the user and the world set, and they constrain both outputs. STEP has two parts: $a_t = \text{ACT}(z_t, x_t; \theta_t, c_t)$ returns the chosen action, and $z_{t+1} = \text{UPDATE}(z_t, x_t; \theta_t, c_t)$ returns the state to carry into the next step. ACT and UPDATE correspond to the two core properties of the continuing loop: ACT to *operational continuity*, and UPDATE to *state durability*.

Operational continuity. The assistant can decide whether and when to act without a new explicit request. Let $\mathcal{T}$ denote all steps and $\mathcal{R}$ those that carry out an explicit request as asked. In a request-only assistant, every step lies in $\mathcal{R}$, while *operational continuity* requires steps outside $\mathcal{R}$ at which ACT decides whether to act:

$$\pi_i^{\theta_t}(\cdot \mid I_{i,t}; c_t) \text{ selects an operation at every } t \in \mathcal{T}, \qquad \begin{cases} \mathcal{R} \subsetneq \mathcal{T} & \text{operational continuity,} \\ \mathcal{R} = \mathcal{T} & \text{request-only.} \end{cases} \tag{1}$$

At each step, each module i among Sensing, Memory, and Action chooses an operation through its local policy π_i. ACT returns the operation that Action's policy π_A chooses. The policy uses a decision context $I_{i,t}$ that combines external input, relevant local state, and information from incoming channels, under the shared conditions c_t. Steps outside $\mathcal{R}$ include the assistant's own checks and events that no request covers, such as an inbox scan for urgent mail or a notice that a flight is delayed. A daily alarm, by contrast, runs around the clock but only follows the user's setting, so all of its steps lie inside $\mathcal{R}$.

The reminder assistant of §1 shows where a step outside $\mathcal{R}$ arises. The conversation has ended, but the reminder remains a pending commitment that the user requested. Later, the assistant detects the end of the meeting, for example from a calendar event. This detection only tests the condition the user set, so it stays inside $\mathcal{R}$. The user, however, left the timing to the assistant, so choosing when to send the reminder happens at steps outside $\mathcal{R}$. At each of these steps, Sensing checks whether the user is free, Memory supplies the pending reminder, and Action decides whether to remind now or wait. Doing nothing is also a valid choice for each module: Sensing can skip an observation, Memory can leave the records unchanged, and Action can stay silent.

State durability. Relevant information remains available after an interaction ends. Let $\mathcal{B}$ denote the steps that end an interaction, such as closing a conversation or fulfilling a commitment. At these boundaries, the assistant updates its running state instead of automatically resetting it to the initial state z_0:

$$\begin{aligned} \text{state durability:} \quad & z_{t+1} = \text{UPDATE}(z_t, x_t; \theta_t, c_t), && t \in \mathcal{B}, \\ \text{session-local:} \quad & z_{t+1} = z_0, && t \in \mathcal{B}. \end{aligned} \tag{2}$$

State durability requires UPDATE to keep the part of z_t that is still valid. The running state z_t holds current context, retained content, task progress, and pending commitments. The mechanism θ_t determines which of these parts remain valid and stores them, and the shared conditions c_t limit what it is allowed to store. A pending commitment, for example, remains in z_{t+1} after the conversation that created it ends, while a privacy requirement can forbid storing that conversation's transcript.

2.1.2 Self-Evolving Extension

A continuing loop can retain information across interactions while its mechanism stays fixed. A *self-evolving extension* goes a step further and uses operating experience to make durable revisions to this mechanism, which later steps then use. Following the definition of a self-evolving strategy by Gao et al. [42], we write each revision as

$$\theta_{t+1} = \text{REVISE}(\theta_t; \tau_t), \tag{3}$$

where the trajectory τ_t holds the assistant's inputs, actions, and feedback up to step t, which form its operating experience. Two features set self-evolution in an always-on assistant apart from this task-based definition. Revisions can happen at any runtime step, including between requests, instead of after each task. The running state z_t also stays outside θ_t, whereas the agent system in that definition includes its memory. In a self-evolving loop, REVISE changes θ_t as this experience accumulates over the user's long-term use, through the user's feedback, the assistant's interactions with the user, and the outcomes of its actions. A revised mechanism then stays in effect across interaction boundaries until a later revision replaces it. These revisions aim to improve later assistance. We count a change as a mechanism revision when it edits a part of θ_t, such as a rule, a prompt, a skill file, or model weights, and so changes how the assistant decides. A change that only adds or corrects content in the running state z_t, such as storing that the user prefers reminders at their workstation, is a state update. Editing the timing rule to wait until the user is back there is instead a mechanism revision, although both changes delay the next reminder. The two differ in the evidence and

controls they need. A state update is a record that the unchanged mechanism reads, and the user can inspect, correct, or delete it. A mechanism revision changes how every later input is handled, so it has to be traced to the evidence behind it, checked before it takes effect, and kept reversible.

A claim of self-evolution therefore requires evidence both that the mechanism changed durably and that later steps used the revised mechanism. When a revision uses outcome feedback, the claim rests on a chain of evidence beginning with retained action records. By the definition of REVISE, this chain has five steps: record, outcome, attribution, revision, and later use. In Eq. (3), the trajectory τ_t must hold the record of an action and its outcome, REVISE must attribute that outcome to the decision behind it and change θ_t, and later steps must use the revised θ_{t+1}. In an always-on assistant, the record must survive until the outcome arrives, often after the interaction has ended, and the outcome must be tied to one of several earlier decisions. The change must also reach θ_t instead of only the running state, and the revised mechanism must carry across interaction boundaries into later decisions. In the reminder example, the assistant records that it sent a reminder when the meeting ended, and the outcome is that the user dismissed it. Attribution ties the dismissal to the timing rule behind that reminder, the revision changes the rule to wait until the user is back at the workstation, and later use is the next reminder sent under the revised rule.

An always-on assistant continues to use a revised mechanism across interactions, so the shared conditions c_t should constrain both its ongoing operation and the revisions that shape its later behavior. We call self-evolution *governed* when these conditions also constrain REVISE, as in $\theta_{t+1} = \text{REVISE}(\theta_t; \tau_t, c_t)$. A governed revision must work within the limits these conditions set without changing those limits itself. Privacy requirements therefore restrict which records it is allowed to learn from, while the user's authority determines what it is allowed to change and when it takes effect. Within these limits, the assistant can attempt revisions when new operating experience warrants a change or at scheduled times. Revisions can also reinforce a mistake. We call this the *memory trap*: faulty memory biases how feedback is interpreted, so later revisions preserve or amplify the original error (§4.4.3).

In general, our formulation of the continuing loop and its self-evolving extension draws on reinforcement learning: z_t corresponds to the agent state and θ_t to the policy parameters. Context-driven mobile-health interventions illustrate *operational continuity* by deciding whether to offer support without requiring a new user request [63]. These decisions can also depend on earlier interactions. For example, a policy learned from mobile-health data uses past notifications to decide whether to send another [64]. This past information is what *state durability* keeps available. Beyond retaining information, reset-free learning [65] studies agents that keep learning during ongoing operation. This provides a precedent for our *self-evolving* extension, in which REVISE changes θ_t for use in later decisions. This continuing perspective also motivates evaluation over time. Long-run average reward provides a basis for evaluating sustained net utility [64]. Long-horizon recommendation considers how recommendations affect engagement in later sessions [66, 67], the kind of delayed outcome that always-on assistants also face.

2.2 Functional Modules: Spaces and Local Policies

We then decompose the assistant into three functional modules: *Sensing*, *Memory*, and *Action*. Several prior studies inform this decomposition. Soar [68] distinguishes perception, memory, and action selection, and CoALA [69] organizes agents built on language models around memory, internal and external actions, and decision-making. Active perception [70] treats information acquisition as a controlled choice, and internal-model accounts [71, 72], in which an agent keeps a model of the world, explain anticipation. In our formulation, each module has a local policy that keeps running between requests, and we study the directed channels between modules. Prediction is not a fourth module in this decomposition, because each local policy already predicts. Sensing anticipates when an event will occur, Memory anticipates what a later decision will need, and Action anticipates what follows from acting now or waiting. A predictive or world model that supports these anticipations belongs to the implementation (§2.4) and can serve all three policies, as a shared language model does. Keeping prediction inside each policy also shows which module's decision a wrong prediction misleads, and the channels trace exactly this dependency.

Sensing. Sensing acquires and interprets current information to provide context about the user, environment, and execution progress. We classify sensing by how it starts (§3). *Continuous* sensing stays active, *triggered*

sensing starts on a cue, timer, or event, and *on-demand* sensing starts when a user request or pending decision needs fresh information. All three draw on sensors, user input, application events, and tool responses within and between requests, and they preserve the time, source, and uncertainty of what they observe.

Memory. Memory maintains information about the user and their tasks across interactions and makes it available for later decisions. We classify records by the time they describe (§4). *Retrospective* memory records the past: events, interactions, and the assistant's actions and outcomes. *Profile* memory describes current facts, preferences, goals, plans, and task progress. *Prospective* memory looks ahead and retains unfulfilled commitments.

Action. Action decides whether, when, and how the assistant helps. We classify actions by what starts them (§5). *Reactive* action responds to a current request, and *deferred* action carries out an earlier commitment without a new request. *Proactive* action addresses a need not covered by an existing request or commitment. Across all three classes, Action can remain silent or act, using planning and anticipated consequences to guide multi-step decisions [32]. Action context encompasses the assistant's intent, the rationale behind its choice, its execution trace, pending commitments, and resulting outcomes.

Spaces and local policies. A module's *space* defines what the module can do and represent, that is, the options it has. It has two parts: the *classes*, as introduced above, and their *format*. The format describes each class concretely: what an observation carries, how a record is represented and stored, or through which interface an action takes effect. The *local policy* selects among these options under the shared conditions. Acting and staying silent, for example, are both options in the action space, and the rule that chooses between them is the policy.

2.3 Six Coupling Channels

Definition: Coupling channel

A *coupling channel* is a directed information flow at the functional level: one module supplies information as context for another module's decisions or state updates. It identifies the source, the supplied context, and how the receiving module uses it. Three modules give six possible directed channels.

We model these channels to trace how information from one module supports decisions in another. This lets us compare systems by their information dependencies and locate gaps that can disrupt later assistance. These flows can then be tested separately or together to assess their contribution. To describe these flows, we write $y_t^{i \to j}$ for the information that module i supplies to module j at step t, which becomes part of the receiver's decision context $I_{j,t}$ in Eq. (1). Table 2 summarizes the three modules, the six directed channels, and their main challenges.

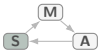
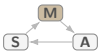
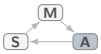
Running example: a reminder after a meeting. Consider a user who asks for a reminder after a meeting. Sensing passes the interpreted request to Memory through $\mathrm{S} \to \mathrm{M}$, where the pending reminder is retained. During the meeting, retained meeting information guides Sensing through $\mathrm{M} \to \mathrm{S}$, helping it decide when to check whether the meeting has ended. Around the expected end time, Action requests a fresh check of the user's availability through $\mathrm{A} \to \mathrm{S}$. Sensing returns the current context through $\mathrm{S} \to \mathrm{A}$, while Memory supplies the reminder and relevant preferences through $\mathrm{M} \to \mathrm{A}$. Together, these inputs help Action decide whether to remind the user or wait. Action then sends the decision and execution context to Memory through $\mathrm{A} \to \mathrm{M}$, so that the task and its history reflect what happened. If the user later reports that the reminder came too soon, the feedback reaches Memory through $\mathrm{S} \to \mathrm{M}$, and once it is attributed to the reminder decision, it can revise when Action sends later reminders.

2.4 Shared Conditions and Implementation

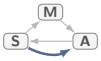
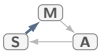
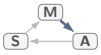
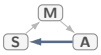
Shared conditions. Shared conditions c_t are the limits set by the user and the world, and they span privacy requirements, user authority, safety, resource budgets, availability, and uncertainty. Privacy requirements

Table 2 The framework: three modules and six directed channels. Space entries summarize the classes introduced in §2.2, and local policies describe what each module decides. Channel entries identify the information supplied and how the receiving module uses it. Miniature loops locate the modules and channels.

A. Functional modules

Module	Space	Local policy	Core challenges
Sensing 	Continuous, triggered, and on-demand observations	Chooses when and what to observe, and how to process observations into useful context.	Missed events are hard to check. Resource and privacy limits also make policy improvements hard to verify.
Memory 	Retrospective, profile, and prospective records	Chooses what information to keep, how to keep it current, and when to make it available.	Wrong or outdated records are hard to detect. Even after correction, their effects on decisions and policy revisions need checking.
Action 	Reactive, deferred, and proactive actions	Chooses whether, when, and how to act.	Repeated help burdens the user, while service pauses risk missed or repeated work. Late or limited feedback makes lasting benefits and policy improvements hard to judge.

B. Coupling channels

Channel	Information flow	Difficulty*	Core challenges
S→A 	Current context and observed execution state that inform Action's decisions.	**Basic**	Context must reach Action in time and remain current when used.
S→M 	Observations that Memory uses to create or update records.	**Moderate**	Future needs are hard to predict. Losing the original evidence also makes records harder to check and correct.
M→A 	Retained personal information and pending commitments that inform Action's decisions.	**Moderate**	Records must be relevant and up to date. Their use must also respect current permissions.
A→S 	Action plans and execution state that guide what Sensing observes and checks.	**Moderate**	Delayed or partly visible effects are hard to check. Checks must also follow plan changes.
M→S 	Retained information that guides what Sensing observes and how it interprets new observations.	**High**	When outdated records guide Sensing, evidence needed to correct them is harder to collect.
A→M 	Action context and immediate results that Memory uses to maintain task history.	**High**	When several actions affect an outcome, linking late feedback to the right one requires earlier decision records.

* Ratings reflect our qualitative assessment of channel difficulty (§§3–5). High indicates errors that are hard to detect and correct.

limit the use of personal information: a confidential meeting can permit transient speech processing while prohibiting transcript retention or disclosure through tools. User authority means the permissions that the user has granted and that are still in force, as when a calendar grant permits reading but not editing. Safety rules out harms that no grant can make acceptable, such as a robot arm moving while a person is within reach or health advice that could injure the user. Resource budgets limit the battery, computation, storage, and user attention that the assistant can spend. Availability concerns whether the devices, stores, and interfaces a decision needs can be reached at that moment. Uncertainty concerns how far the assistant can trust its current evidence, such as whether the meeting has really ended.

Implementation: models and scaffolding. The implementation consists of the model parameters ω_t and the scaffolding Σ_t around the model. The model, typically a large language or multimodal model, interprets the input and makes decisions, while the scaffolding orchestrates it and carries its outputs through interfaces to the user's devices, applications, and databases. Both act on every module: they take in the information

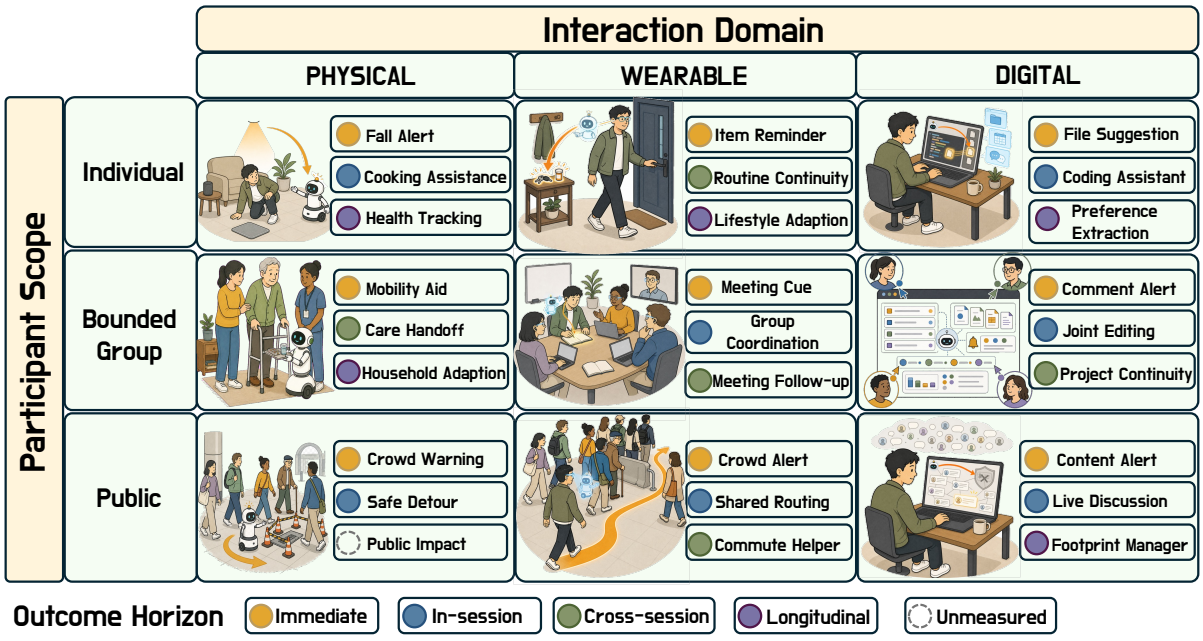


Figure 4 Three dimensions of application scope: interaction domain (where observations and actions occur), participant scope (who is involved or affected), and outcome horizon (when consequences become observable). Each cell shows selected examples within that setting.

that a module receives and process it into a form that the next module can use. The modules are therefore functional divisions, not separate models or tools, and this survey organizes the literature by function. The details of implementation engineering, such as context engineering [73] and the design of self-improving [43] and long-horizon agents [60], are outside the scope of this survey.

2.5 Application Scope and Adjacent Concepts

As shown in Figure 4, we describe the application scope of an assistant by three coordinates. The *interaction domain* denotes where observations and actions occur: physical, wearable, or digital, though an assistant can span several. The *participant scope* denotes who is involved or affected: an individual, a bounded group, or the public. The *outcome horizon* marks when the evaluated consequence becomes observable: immediately, within a session, across sessions, or over a longitudinal study that follows the same participants for a long period. Missing measurements are marked *unmeasured*.

Adjacent concepts. Table 3 compares adjacent kinds of agent by *operational continuity*, *state durability*, and mechanism revision. The three questions below clarify these relationships.

Q1: How does always-on assistance differ from long-horizon agency?

Long-horizon agents coordinate dependent steps toward given tasks, across sessions or even an open-ended stream of tasks [60]. Always-on personal intelligence concerns how assistance continues as a user's goals change or new ones arise, including periods without requests.

Q2: Why require both operational continuity and state durability?

Operational continuity lets the assistant observe and act after an interaction ends, and state durability lets it carry what it has learned and promised into the next one. With durability alone, an assistant remembers the user but only reacts to requests [18, 19], while with continuity alone, it can act proactively but only within one interaction [12, 74]. When both are present, they depend on each other: retained state guides

decisions made between requests, while new observations and action outcomes determine what remains in that state. This mutual dependence makes continuing assistance a distinct unit of analysis and evaluation: assessments must look beyond single turns to track missed commitments, unnecessary interruptions, and cumulative resource costs across extended interactions.

Q3: How does self-evolution relate to always-on assistance?

Self-evolution extends the always-on assistance loop toward the user's needs, the direction in which the assistance grows more personal. Research on self-improvement studies how experience can refine models or their scaffolding to improve future performance [42, 55], using feedback such as environment rewards or model critiques [43]. The challenge is to turn the experience of daily use into durable adjustments, establish whether they improve future assistance, and preserve ongoing commitments and user authority while adapting. In its strong sense, recursive self-improvement goes a step further: it also revises the process that produces these revisions, so that each round can change how the next round improves [75]. In an always-on assistant, such recursion is hard to implement and to evaluate, and few of the works we review attempt it. We therefore leave it to future work and state it only as an open problem in §8.3.

Table 3 Always-on assistants and adjacent kinds of agent. The first two properties define a continuing loop. The third, durable mechanism revision from operating experience, yields a self-evolving loop only on top of both (§2.1). ✓ = yes; ✗ = no; ○ = not required. Rows describe typical designs, and a system can combine them.

System type	Operational continuity	State durability	Mechanism revision
Session-local conversational assistant	✗	✗	✗
Request-only assistant with durable memory	✗	✓	✗
Long-horizon agent	○	○	○
Self-improving agent	○	○	✓
Streaming assistant without durable user state	✓	✗	✗
Scheduled or background assistant acting as set	✗	○	✗
Always-On Personal Intelligence			
Continuing loop	✓	✓	○
Self-evolving loop	✓	✓	✓

2.6 Mapping the Literature

Figure 5 maps software and hardware developments onto the framework, and both show the same shift: from single capabilities, to systems that connect Sensing, Memory, and Action, and then toward systems that revise themselves. Early software systems built each function separately around a hand-designed source of context. Cyberguide [76], StartleCam [77], and SenseCam [78] captured context for Sensing, Memex [79], the Remembrance Agent [80], and Stuff I've Seen [81] organized personal information for Memory, and Lumière [11], CALO [82], and BusyBody [83] used context to decide when to assist. More recently, learned models made each module general across inputs and tasks, and research advanced the modules on separate tracks that expand their options, choose among them, or evaluate them. Screen parsing and streaming speech models let Sensing read app and web screens and follow continuous sound [84, 85, 86, 87, 88, 89, 90, 91], external memory let Memory outgrow the model's context [19, 92], and work on computer use and response timing extended Action to open software and to the choice of when to respond [93, 21]. Most recently, systems connect the three modules in wearable, digital, and physical applications [8, 94, 95, 96, 2, 97], and the information that passes between modules becomes a difficulty of its own. Each module's capability still limits the loop, since streaming video models, for example, struggle with the order of events and with fine-grained actions (§3.5.2). An emerging trend is that systems also revise their own mechanism, for example by editing their own code [98, 99], which moves the target of improvement from the records about the user to the assistant itself.

Hardware has followed a similar path, moving sensing and computation closer to the user. Early sensor-driven devices added observation sources such as depth and low-power motion sensing [100, 101], while wearable

prototypes carried capture and display into daily activity [78, 7, 102]. More recently, handheld and extended-reality devices, which mix digital content into what the user sees, pair perception with large language and vision models, so the device itself can interpret a scene and assist [103, 104, 105]. Most recently, humanoid robots connect onboard models to manipulation, extending the loop from advice to physical action [106, 107].

Requirements and Evidence for Always-On Assistants

Core requirements. An always-on assistant must satisfy two properties, (1) *operational continuity* and (2) *state durability*. Under the first, it uses current context to decide whether and when to act without a new user request, with waiting as a valid choice. The second requires it to retain usable personal state across interactions for later assistance.

Implementation and evidence. Developers of an always-on assistant should describe its intended use, shared conditions, and implemented coupling channels. Not every assistant needs all coupling channels, but for each implemented channel, developers can show what information it carries and how the receiving module uses it. Tracking user outcomes over time then shows the benefits and costs of these design choices.

Self-evolution as an advanced extension. Self-evolution extends the continuing loop toward the user's needs, revising the mechanism from operating experience so that the assistance grows more personal. A claim of self-evolution should show that the revision persists and is used in later steps, and it should separate mechanism revisions from updates to personal content. When a revision uses outcome feedback, the evidence traces the five steps: record, outcome, attribution, revision, and later use. Governed self-evolution also requires the shared conditions to constrain revisions, and a claim of improvement should show greater benefit to the user than the earlier version under comparable conditions and over the relevant horizon.

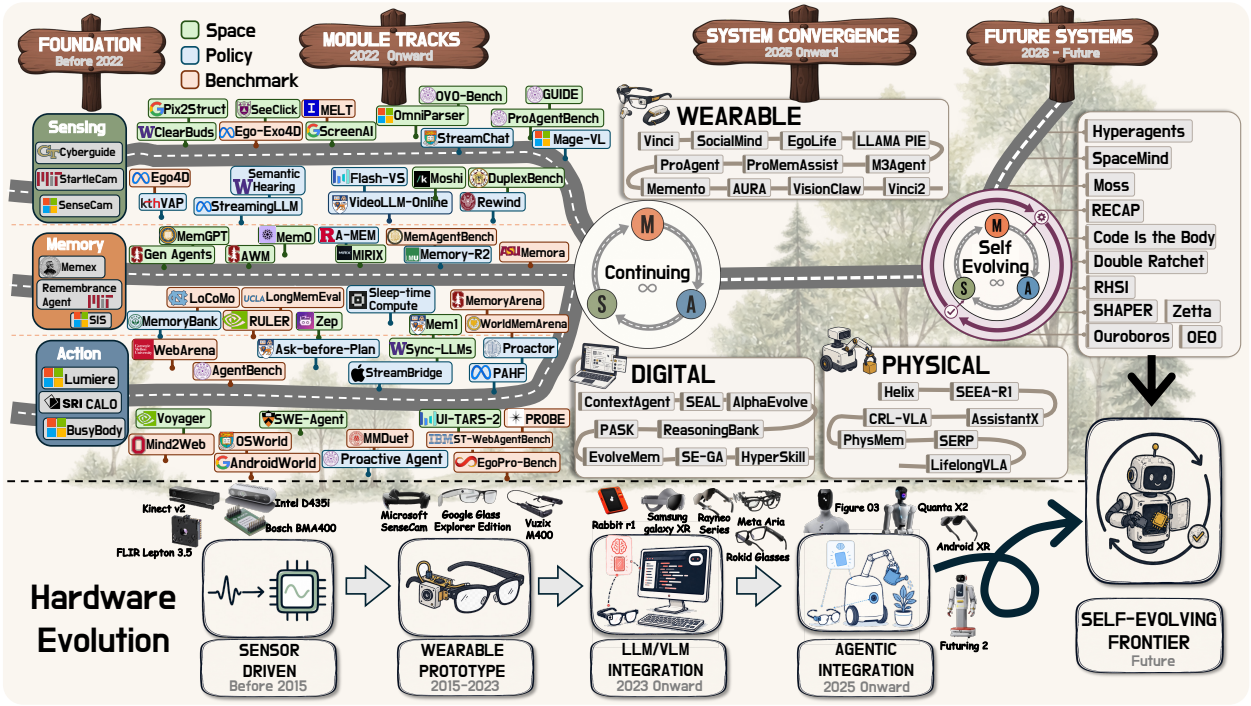


Figure 5 From module development to system convergence. We show the convergence trend of always-on systems. In both software and hardware, development moves from single capabilities to integrated systems that connect Sensing, Memory, and Action, and then toward self-evolving systems.

3 Sensing

Sensing takes in observations and feedback from the user and the world, forming the current context and the state of ongoing tasks. In an always-on assistant, sensing continues between requests, so the assistant decides for itself when to switch observation on and what to look at. It has to choose because three limits keep it from observing everything all the time. First, energy: on a battery-powered device, observation shares one budget with memory and action. Second, privacy: a camera or microphone also records people who have not agreed to it. Third, time: interpreting what is captured takes processing, so evidence can arrive too late for the decision that needs it. Sensing that stays on sees the most but spends the most energy and records the most bystanders. Waking it only on a cue saves energy but misses events that give no cue, and starting it only on request costs the least but makes a decision wait for capture and interpretation. Beyond these trade-offs, Memory tells sensing what to look for and how to read it, and Action asks it to check what an action changed. As shown in Figure 6, we describe the sensing space, then cover shared conditions, the local policy, coupling, and evaluation, and finally discuss the challenges that remain for sensing.

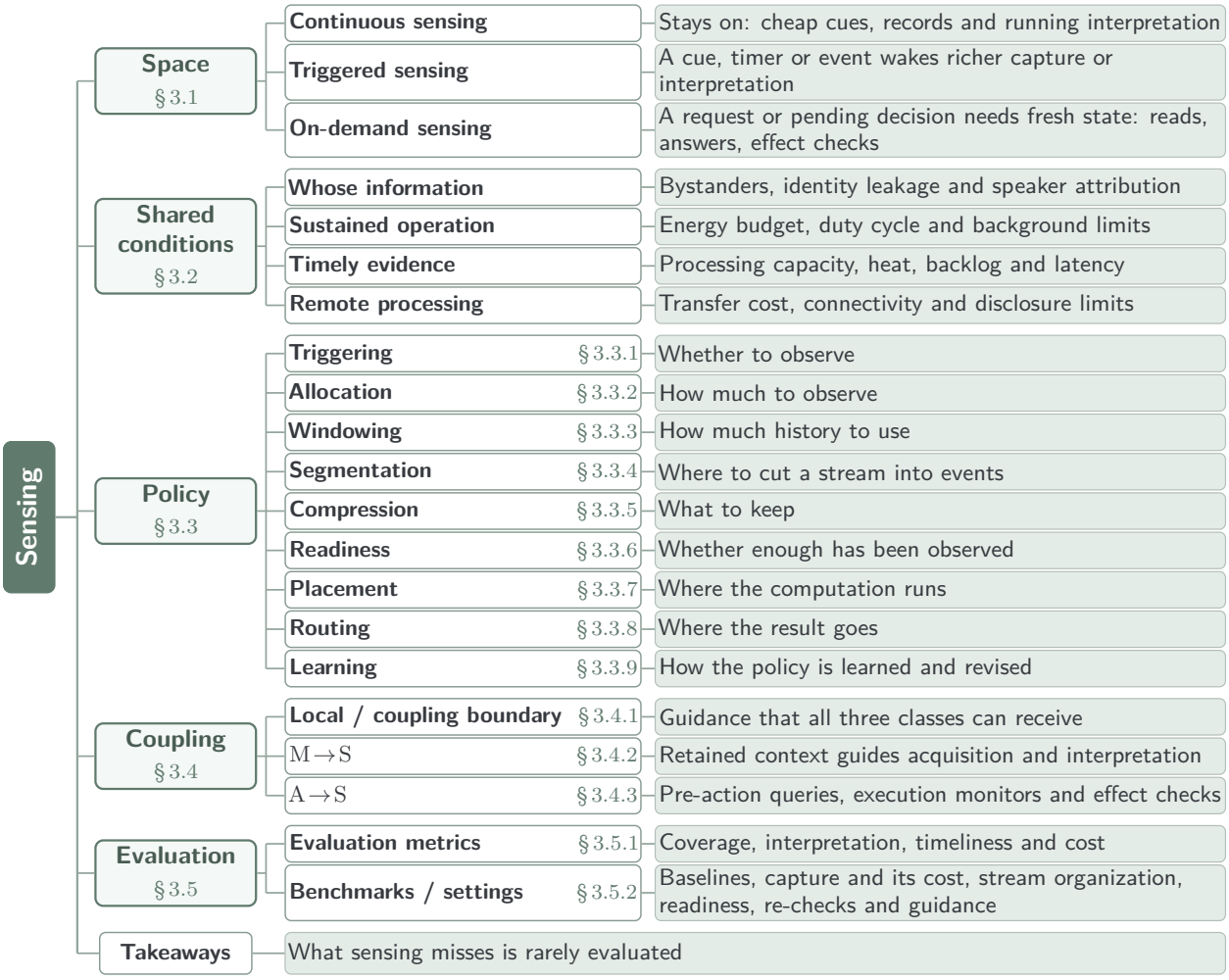


Figure 6 Sensing section structure and taxonomy.

3.1 Sensing Space

The sensing space covers the current information available to the assistant, from observations to the current state built from them. Selecting and interpreting observations provides evidence for later decisions [108, 70, 109]. Following the two parts of a module's space, §3.1.1 classifies sensing by what starts it, into **continuous**,

triggered, and **on-demand** sensing, and §3.1.2 describes the format of observations: the forms their content takes and the source, time, uncertainty, and permissions they carry.

3.1.1 What Starts Observation

Continuous sensing. Table 4 lists the different classes of sensing. Continuous sensing keeps observing whenever the device can sense, without waiting for a cue or a request. We divide it into three groups by what it supplies. First, **cue detection** is the cheapest: voice-activity and keyword chips run at nanowatt to microwatt power [110, 111]. Personal VAD [112] narrows voice detection to the enrolled user, whose voice was registered in advance, and runs on the device. EgoCHARM [113] uses movement instead, recognizing activity from motion sensors on a low-power coprocessor. Sensor foundation models are trained on large and varied sensor data, and they address how such cues transfer across users, devices, and contexts [47]. Second, **record keeping** stores what is observed: ChainStream [114] treats sensor readings, screen captures, and messages as streams processed as they arrive. Activity Frames [115] gives such records more structure, organizing passively captured desktop activity into timed episodes linked to the original evidence. EgoLife [94] widens their coverage, extending egocentric recording to multi-day, multi-person activity, following earlier personal archives [116]. Third, **running interpretation** keeps a model interpreting video or speech. VideoLLM-online [74], StreamingVLM [117], and Mage-VL [118] process video frames as they arrive. For speech, VibeVoice-ASR-Streaming [119] delivers speaker-attributed text while audio continues, and Moshi [91] keeps receiving speech while it replies. For video, this running interpretation assumes a camera that stays on, which current glasses sustain only for short periods [120].[1] The three groups thus trade cost against detail: cue detection is cheap but coarse, while record keeping and running interpretation give richer evidence at a higher cost. Overall, this class keeps the context current for the decisions made between requests, and triggered sensing, described next, starts richer capture only when a cue calls for it.

Triggered sensing. Triggered sensing wakes richer capture or interpretation when a cue, timer, or event occurs, and its three groups differ in what raises the cue: a sensor or timer, a small model reading captured data, or a condition that another module registers. First, **wake-up triggers** act before capture or computation, and timers are the simplest: SenseCam [78] took photographs on a timer and when its light and infrared sensors changed. On current wearables, the cue is often another sensor: EgoTrigger [121] classifies hand–object interaction sounds to trigger image capture, a design aimed at all-day smart glasses. OpenGlass [122] applies the same idea to computation, using low-power wake-up sensing to activate an inference processor that is powered down between inferences. Second, **model gates** filter data after it is captured, using a small, fast model to decide whether to wake a larger, more expensive one. For instance, an audio detector can spot likely actions so that a vision–language model runs only on those moments [123]. Similarly, PRPF [124] uses a small multimodal model to filter incoming streams before it calls a larger reasoning model, and the same filter decides when its assistant intervenes. StreamMind [125] and Event-VStream [126] move this filter inside the model. Their video encoders process every frame, but the language model runs only when StreamMind's learned event gate opens or when Event-VStream detects a transition between events. Cambrian-S [127] uses prediction error as the cue instead and interprets accumulated video when the observed frames diverge from its predictions. Third, **registered triggers** are set by another module from a pending request: Em-Garde [128] and the agent in IPIBench [129], for example, keep a user's pending monitoring request, such as a notification when the water boils, and check each new stretch of video against it. Triggered sensing keeps observation going between requests at lower cost, and its trigger rules are among the parts of the mechanism a self-evolving loop could revise. However, a trigger misses changes that give no monitored cue, so a silent booking cancellation remains undetected until a scheduled or on-demand check.

On-demand sensing. On-demand sensing starts when a user's request, or a pending decision that needs fresh state, calls for it, and its three groups differ in what is read: the state a decision needs, the user's answer, or an action's effect. First, **targeted reads** are the most common and can follow a predicted need, a task that needs more detail, or a user's request. ProAgent [130], for instance, combines low-cost contextual cues with richer perception on demand and reads agendas, weather, or transit information once it

[1] Aria Gen 1 hardware specifications: https://facebookresearch.github.io/projectaria_tools/docs/tech_spec/hardware_spec

predicts that help is needed. ContextAgent [96] likewise calls tools for the context that a predicted need still lacks. Alpha-Service [131] invokes a streaming video model only when a task needs finer visual detail. PULSE [132] lets agents choose which smartphone-sensing modalities and time windows to inspect instead of receiving a fixed summary. In the home, SAGE [133] reads device information as part of the actions that answer a request. Some decisions also need to know that retained state has gone stale. SpeechLess [134], for example, refreshes remembered answers about live content such as weather or transit before reusing them. Second, **answer interpretation** reads the user's responses or feedback to the assistant. PrISM [135], for instance, refines its step estimate from a reply. Third, **effect checks** read what an action changed, as when Mobile-Agent-E [136] compares screens before and after each action. On-demand sensing supplies fresh state to user requests and also to decisions that no request starts, and its checks after execution return the outcomes that feedback-driven revisions need.

3.1.2 Observation Formats and Metadata

Each class of sensing above delivers its result as observations, and their format decides what Memory and Action can later do with them. An observation reaches the rest of the assistant in one of five forms, which differ in how much detail they keep and in which models or modules can read them. Which form to keep is a policy choice that we discuss in §3.3.5.

Raw signals and learned features. Raw signals keep every captured audio sample and video frame. Any model can interpret them again later, but they cost the most to store and expose the most private content. Learned features are the vectors that a trained model computes from each frame or stretch of audio. They take less space, but only the model that computed them can read them. The KV cache [159] is one such store, and systems such as StreamingLLM [160] and Cartridges [161] differ in how long they keep it. Stored features, however, must be computed again from the raw signal once the assistant switches to another model, and that signal may no longer be kept.

Compressed video. Compressed video, which lies between raw signals and learned features, suits always-on sensing. A phone or camera produces it anyway whenever it records, it is far smaller than the raw frames, and any decoder can turn it back into frames, so unlike learned features it is not tied to one model. An emerging line of codec-based models reads this stream directly. The standard H.264 and HEVC codecs store most frames as motion vectors, which record how each block of the image moved from a reference frame, and residuals, which record what that movement does not explain. These models use the two signals in two ways. The first uses them as they are, at almost no extra cost. Each patch, a small block of the frame, is scored by its motion and residual, and between key frames, which the codec stores in full, only the patches that changed most are encoded [162, 163, 118]. The second trains an encoder to read these signals directly, so the frames between key frames enter the model as compact features instead of full images [164, 165]. CoPE-VideoLM [165], for example, uses about a fifth of the tokens of a 64-frame baseline at comparable accuracy and cuts the time to the first output token from 2.39 s to 0.33 s. Other frame selection and token pruning methods also save tokens, but they work on decoded frames without the codec's signals, so we discuss them in §3.3.2.

Text and structured items. Text and structured items differ from these three forms in that other modules can read them directly. A transcript or a caption puts what was heard or seen into words, and an interface element or a timed event arrives with named fields. In exchange, both keep only what was selected when they were produced, and a detail left out at that point cannot be recovered from them later. This trade-off becomes visible when one source is observed in two forms. For example, a web page on screen can be captured as a screenshot or read through its structured items. These include the document structure and the accessibility metadata, the element labels and roles that the page provides for screen readers. The screenshot supplies visual information that this structure lacks. However, the structure can go stale when the page changes after it was read, so the two forms disagree. Zhang et al. [166] find that models then often follow the structure, even though they read the same screenshot accurately when it is shown alone. Such disagreements matter for recurring observation, which checks the same source again and again. Each check relies on compact inputs to

Table 4 **Representative sensing methods by what starts observation.**

Work	Group	Key feature	Input	URL
Continuous sensing				
VibeVoice-ASR [119]	Running interpretation	Incremental attributed text	Streaming audio	/GH
StreamingVLM [117]	Running interpretation	Short visual, long text spans	Streaming video	/GH
Mage-VL [118]	Running interpretation	Reads compressed video	Streaming video	/GH
Personal VAD [112]	Cue detection	Target-speaker conditioning	Audio + voiceprint	/–
EgoCHARM [113]	Cue detection	Activity from motion, low power	Egocentric IMU	/–
ChainStream [114]	Record keeping	Process arriving events	Sensor + app streams	/GH
EgoLife [94]	Record keeping	Multi-day, multi-person recording	Glasses video + audio	/GH
LightMem-Ego [137]	Record keeping	Background capture, later recall	Phones and glasses	/GH
Omi [138]	Record keeping	Action items from conversations	Wearable audio	/–
VideoLLM-online [74]	Running interpretation	Frames processed on arrival	Streaming video	/GH
Moshi [91]	Running interpretation	Listens while it replies	Full-duplex speech	/GH
M3-Agent [95]	Running interpretation	Faces and voices to identities	Video + audio stream	/GH
Proactive Agent [12]	Cue detection	Activity events signal a need	Computer event streams	/GH
Activity Frames [115]	Record keeping	Timed episodes + sources	Desktop screen frames	/GH
Windows Recall [139]	Record keeping	Searchable screen history	Desktop screen	/–
Kirigami [140]	Cue detection	Remove speech segments	Ambient audio	/–
N_0-TWAM [141]	Cue detection	Contact events mark task phases	Tactile + vision	/GH
DomusFM [142]	Cue detection	Pretrained on home sensor events	Smart-home sensors	/GH
ReMEmbR [143]	Record keeping	Captions stored with pose, time	Robot video + pose	/GH
Triggered sensing				
Cambrian-S [127]	Model gate	Prediction error as cue	Streaming video	/GH
Dispider [144]	Model gate	Decision module apart from replies	Streaming video	/GH
SenseCam [78]	Wake-up trigger	Capture on sensor change	Timer / light / infrared	/–
EgoTrigger [121]	Wake-up trigger	Audio-triggered capture	Audio + camera	/GH
OpenGlass [122]	Wake-up trigger	Sleeps between inferences	Wake sensor + camera	/–
Pro^{2}Assist [145]	Wake-up trigger	Head motion selects key moments	AR glasses	/–
StreamMind [125]	Model gate	Learned gate before LLM	Streaming video	/–
Acoustic Triage [123]	Model gate	Audio selects moments for VLM	Audio + video	/GH
Event-VStream [126]	Model gate	Transitions resume generation	Streaming video	/–
GazeAgent [146]	Model gate	Behavior cues gate recalibration	Gaze + IMU + video	/–
Em-Garde [128]	Registered trigger	Query compiled to proposals	Streaming video	/GH
IPIBench [129]	Registered trigger	Checks stream against requests	Streaming video	/–
PRPF [124]	Model gate	Light perceptor gates reasoner	Multimodal stream	/–
TGL controller [147]	Model gate	Graph model decides when to wake	User activity events	/–
HomeEmergency [148]	Wake-up trigger	Emergency sound starts search	Audio + robot camera	/–
Event-Driven Assist [149]	Model gate	State change invokes VLM	Tabletop RGB-D	/–
Code-as-Monitor [150]	Registered trigger	Constraints as visual checks	Robot cameras	/–
On-demand sensing				
ContextAgent [96]	Targeted read	Tools for a predicted need	Sensors + tools	/GH
ProAgent [130]	Targeted read	Reads apps on predicted need	Phone sensors + apps	/–
Alpha-Service [131]	Targeted read	Calls video model as needed	Glasses video	/–
PULSE [132]	Targeted read	Picks modalities + windows	Phone sensing logs	/–
SpeechLess [134]	Targeted read	Refreshes live content	Spatial memory + web	/–
PrISM [135]	Answer interpretation	Step estimate from replies	Smartwatch + dialogue	/–
Egocentric Co-Pilot [151]	Answer interpretation	Asks a short clarifying question	Smart glasses	/GH
ProAct [152]	Targeted read	Checks memory before searching	Text chat	/GH
EgoAsk [153]	Answer interpretation	Asks about the user's objects	Glasses + robot	/–
Mobile-Agent-E [136]	Effect check	Compare before/after screens	Phone screenshots	/GH
VeriGUI [154]	Effect check	Verifies each action's effect	GUI screens	/–
Dot tasks [16]	Effect check	Checks results of created tasks	Cloud computer	/–
EyeRobot 2.0 [155]	Targeted read	Gaze moves to what the task needs	Robot stereo eyes	/–
SAGE [133]	Targeted read	Reads device state for a request	Smart home	/–
Vision in Action [156]	Targeted read	Head camera looks past occlusions	Robot head camera	/GH
Explore-EQA [157]	Targeted read	Explores until confident	Robot RGB-D	/GH
MoveCare [158]	Answer interpretation	Confirms alarms before alerting	Robot + smart objects	/–
Hobbit [3]	Effect check	Checks each grasp after acting	Home robot	/–
AssistantX [2]	Effect check	Checks each step's outcome signals	Office robot	/GH
ABot-Claw [6]	Effect check	Critic checks progress online	Robots	/GH

stay cheap, and these inputs must still keep the details needed to notice a change and to settle a conflict with earlier context.

Source, time, uncertainty, and permissions. Besides its content, an observation needs to carry its source, its time, how certain its interpretation is, and who is allowed to use it. Source, time, and certainty matter because the current state combines observations from several sources. A calendar entry, the status of a call application, and speech activity, for example, can each indicate whether the user is free. If the calendar indicates an ongoing meeting while the call application has closed, the state needs to keep both readings with their source, time, and uncertainty, so later decisions can handle the discrepancy. Source and time also let Memory reinterpret a record later. The source also matters for trust, since tool responses can carry untrusted content [167], and external text can redirect the assistant [168]. Who is allowed to use an observation matters because screens and documents opened under the user's login can also contain collaborators' or correspondents' information alongside the user's own content. For such screens, region-level privacy judgments examine which parts a task permits the assistant to use or disclose [169]. Keeping who supplied the information and who is allowed to receive it lets later storage and actions follow those permissions.

3.2 Shared Conditions for Sensing

Whose information is the assistant allowed to observe? Continuous and triggered sensing record whoever is present, not only the user. On wearables, studies of data glasses document context-dependent attitudes toward recording [170, 171] and wearers' difficulty managing bystander privacy [172]. In smart homes, owners and visitors can differ in their preferences about collection, storage, and sharing [173]. Some systems therefore limit what is captured or answered. Kirigami [140] filters speech locally when only non-speech activity cues are needed, and Selective Hearing [174] trains audio models to refuse queries about incidental bystander speech. Limiting capture is not enough on its own, because what the data reveal also depends on the inference. Motion data recorded for activity recognition can reveal the wearer's identity [175], and even a model's internal features can leak it. Tests recover a speaker's identity from the hidden states of two full-duplex speech models, which listen and speak at the same time: Moshi and a SALM-Duplex implementation [176]. The same study measures how anonymizing the waveform or these features affects response quality and latency, so privacy assessment also covers the representations used during processing. Knowing who is speaking also keeps the assistant's own records correct: Personal VAD [112] and online diarization [177], which labels who is speaking as audio arrives, help separate a promise the user makes from another person's remark. As people enter or leave, the assistant reassesses which signals and inferences it captures, processes, or retains.

How long can sensing continue? An always-on assistant needs enough energy to sustain observation under its declared operating schedule, alongside memory access and action [178]. For example, Aria Gen 1 [120] lists 1.5 h of continuous recording, with other sensor settings optimized for 5+ h and 15+ h.[1] Lower sampling can extend recording time but miss short events. On phones, the operating system also limits sensing between requests. Since Android 9, an app in the background cannot use the microphone or camera, and its sensors stop delivering events unless it runs as a foreground service, which keeps a notification visible to the user.[2] We compile reported power measurements and estimates for representative sensing and processing configurations in Figure 7. Wake-up chips consume nanowatt- to microwatt-scale power, while wearable visual processing and phone-based model inference reach milliwatt and watt scales, respectively [110, 111, 122, 179]. The gap between these scales motivates keeping low-cost detection active between requests and starting heavier processing when relevant cues appear.

Can observations reach decisions in time? Continued inference is limited by sustained processing capacity and heat [179]. Throughput is the amount processed per second. For example, in a test of sustained Qwen2.5-1.5B inference on an iPhone 16 Pro, throughput settles 41.5% below its peak [182]. When throughput falls below the rate at which inputs arrive, a backlog can leave Memory and Action using old context even

[1] Aria Gen 1 recording profiles: https://facebookresearch.github.io/projectaria_tools/docs/tech_spec/recording_profiles

[2] Android background sensor limits: https://developer.android.com/about/versions/pie/android-9.0-changes-all

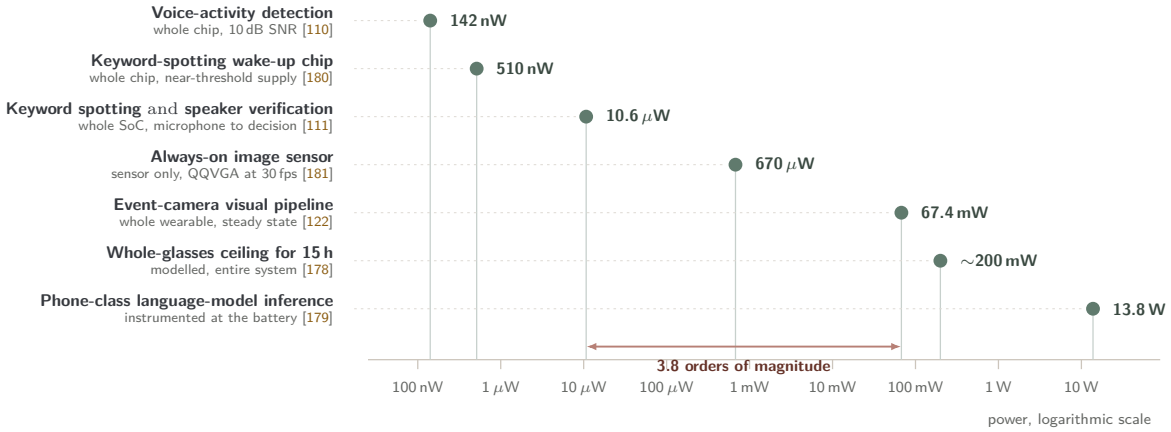


Figure 7 Power costs of always-on processing. Across the reported configurations, audio wake-up uses the least power, always-on visual sensing and processing use more, and LLM-powered processing uses the most, with gaps spanning orders of magnitude. Labels specify the measurement scope; the approximately 200 mW value is a modelled whole-system budget. The configurations illustrate power scales across different tasks and hardware, rather than a controlled comparison of sensing methods.

though capture continues. Acceptable delay also depends on the task: Chen et al. [183] derive target latency ranges of 95–105 ms for continuous billiards guidance and 600 ms–2.7 s for step-by-step assistance. Checking whether the user is free now needs a timely result, whereas a record intended for later review can tolerate more delay.

What does remote processing cost? Offloading sends work to another device and trades local computation for communication: DeepWear [184] shows that transfer overhead can offset faster remote computation. This communication depends on a network, whose delay and disconnection affect when observations become usable. Privacy also constrains what can leave the device, and local filtering is one way to reduce disclosed content [140]. An always-on assistant therefore weighs remote compute against communication cost, connectivity, and permission to transmit evidence.

3.3 Sensing Policy

Whether an event reaches Memory and Action depends not only on the model that interprets it but also on runtime choices about what to capture, keep, and pass on. Table 5 summarizes eight such decisions made by the local sensing policy π_S, from acquisition to delivery. We compare which events each choice captures, what information survives processing, and whether it reaches Memory and Action in time. The policy can combine these choices or reconsider them as new evidence arrives, as when a fixed rule varies the sampling rate with current activity. In §3.3.9, we then discuss how these policies are learned and what revision during deployment would require.

3.3.1 Whether to Observe: Triggering and Wake-Up

Timers and cues. Scheduled capture builds background context without repeated user intervention, while manual control lets the user select a moment but requires their attention. SenseCam [78] supports both, with timer-based photographs, a manual shutter, and start/stop recording. Cues can also set a sampling rate instead of waking a sensor: ProAgent [130] raises its visual sampling rate on motion, location, and conversation cues, and finds that low rates matter little in stationary settings but can miss critical events in dynamic scenes. Across trigger designs, savings depend on the checking cost and activation frequency [123]. False triggers waste capture or computation, and missed triggers can leave decisions using outdated context when no other source records the change.

Table 5 **Local Sensing decisions and representative works.**

Decision	Main choices	Trade-off	Representative works
Triggering	Timer/cue/surprise/request	Misses/checking cost	EgoTrigger [121], Cambrian-S [127], ProAct [152]
Allocation	Rate/resolution/modality/region	Coverage/frame detail	VisualClaw [185], STC [186], Ego-METAS [187]
Windowing	Size/compaction/refresh	Context/processing	StreamingVLM [117], JoyAI-VL [188], StreamChat [189]
Segmentation	Onset/offset/overlap/change	Stability/delay	StartNet [190], ActionSwitch [191], On-GEBD [192]
Compression	Select/summarize/extract/filter	Detail/disclosure	Imprint [193], Kirigami [140]
Readiness	Ready / wait / observe more / flag missing information	Premature use/wait	StreamReady [194], PrISM [135]
Placement	Sensor/device/edge/cloud	Compute/offload cost	EgoCHARM [113], DeepWear [184], Shredder [195]
Routing	Decisions/records/background	Delivery/coordination	Dispider [144], MIRIX [20], Thinking Machines [196]

Requests and pending decisions. Some observations start because a decision needs fresh state. The simplest rule refreshes a class of content every time, as SpeechLess [134] does for live content. Other rules first consult memory, as ProAct [152] does before it searches. A rule can also act on uncertainty in the assistant's belief, meaning its current understanding of the user's situation. CAPTURE [197] keeps this belief as a probability distribution over the user's hidden state. When its uncertainty about how to update memory passes a calibrated threshold, Action asks the user a clarifying question instead of guessing. Evaluations show that such checks and questions are often skipped. In MCB [198], models verify changing facts more reliably than they ask users to resolve ambiguity. For example, Qwen3.5-9B, given only the action definitions, asks on none of 12 clarification items while verifying 12 of 18 freshness items. In VibeLifeBench [199], all seven evaluated frontier models score low on multi-week tasks in which the world changes without announcement. The reviewed evaluations do not demonstrate multi-week learning of when a fresh check justifies its cost.

3.3.2 How Much to Observe: Allocating the Perception Budget

Sampling rate and representation. For a fixed representation, visual tokens per second equal fps multiplied by tokens per frame. Spending more tokens on each frame therefore leaves room for fewer frames under the same budget. Some systems spend a single token per frame: VideoLLM-online [74] pools each frame into one token, and StreamMind [125] combines spatial and temporal features in one perception token per frame. The frame rate matters because many actions are brief. Our calculation from HD-EPIC's [200] released annotations finds that 40.6% of action intervals are shorter than one second, so at a uniform 1 fps such events can fall entirely between successive frames.

Frames, regions, and reuse. General video models choose moments and resolution according to the query [201, 202] or request higher resolution only when needed [203, 204], while on a wearable or phone the device fixes the budget in advance. VisualClaw [185] passes streaming frames through a multi-stage filter before its main model reads them. Compared with sampling eight frames uniformly, with the same skills available, it lowers the cost per question, and at a matched eight-frame budget it still answers more accurately on short-clip benchmarks. Beyond choosing frames, STC [186] reuses and prunes similar visual information during streaming encoding, reducing repeated processing of received observations.

Modalities and task requirements. Across modalities, sensor choice at each step depends on task requirements. Ego-METAS [187] evaluates per-step modality selection under hardware-based energy budgets on continuous egocentric streams. It finds that useful sensor combinations depend on the scenario: learned routing struggles on untrimmed streams, recordings not cut around the events of interest, while simple dynamic fusion offers useful accuracy–energy trade-offs. Between requests, sensor selection balances acquisition cost against the evidence each sensor provides.

3.3.3 How Much History to Use: Observation Windows

Recent context and compacted history. Observation windows bound the history available to interpret the current situation. Window limits discard history, while compaction preserves a less detailed version.

These limits differ in what they keep: StreamingLLM [160] retains initial and recent tokens in a bounded cache, while StreamingVLM [117] uses shorter visual and longer text windows. Flash-VStream [205] preserves detailed recent frames and compacts older observations into temporal and abstract visual representations. JoyAI-VL-Interaction [188] retains recent visual tokens, summarizes earlier chunks as text, and consolidates older summaries at a coarser level. The choice between discarding and compacting controls which earlier instructions and corrections accompany current activity.

Window updates during a response. An observation can reach the model before a response or while the response is still under way. A turn-based model reads its input before it answers, so anything that happens during the answer waits for the next turn. A full-duplex model keeps receiving input while it responds [91, 196, 206]. Receiving new input, however, does not mean using it at once, because systems differ in when the input enters the context they generate from. StreamChat [189] refreshes visual context at each decoding step, that is, for every generated token, allowing new frames to affect an ongoing answer. In contrast, Audio-Interaction [207] continuously encodes audio into a queue. The Audio-Interaction decoder takes in queued input at designated points, such as silence or the end of a response, and generates from the context already received. Refresh timing is thus a choice of its own: while a captured correction remains queued, the decoder continues with the earlier context.

3.3.4 Where to Cut: Online Segmentation

Where a stream is cut decides which observations a later decision can use, and when the cut is detected decides whether it can still help. A boundary that a metric counts as correct can be detected only after the action ends, in time for a record but too late for intervention.

Action progress and onsets. We first consider cutting a stream online based on how far an action has progressed and where it begins. Streaming temporal action segmentation labels arriving video chunks without future observations [208, 209]. OnlineTAS [210] improves such labeling with adaptive context and post-processing that reduce fragmented predictions, and ProTAS [211] adds action progress and learned task graphs, which encode step order, to locate procedural steps. Action-start detection [212, 213, 190] instead locates onset, enabling intervention before completion. Other methods cap the cost of processing a long history: LSTR [214] summarizes older frames coarsely while keeping fine detail for recent ones, and TeSTra [215] reuses past attention computations so that each new frame takes a constant amount of computation.

Overlapping and unfamiliar events. Beyond labeled action steps, other methods address events that overlap or that fall outside named action classes. For overlapping events, ActionSwitch [191] detects simultaneous action intervals without class labels. Two methods center on objects: object-aware egocentric detection [216] uses the objects involved, and Mamba-OTR [217] targets sparse take-and-release events. For events outside named classes, generic event boundary detection [218] locates meaningful changes without naming actions, and On-GEBD [192] decides online. These methods yield different units: concurrent activities, object transitions, or broader changes that prompt reinterpretation.

Boundaries from prediction and change. Event segmentation theory links boundaries to discrepancies between expected and observed events [219]. The prediction error that triggers Cambrian-S [127] also marks its boundaries, and Event-VStream [126] finds its transitions from motion, semantic, and predictive cues. ES-Mem [220] applies event segmentation to dialogue memory. Boundaries thus shape both when processing resumes, as in Event-VStream, and which observations form a retrieval unit, as in ES-Mem.

Boundary timing and its value. The value of a boundary to the assistant depends on when it is detected. StartNet [190] evaluates action starts with offset tolerances from one to ten seconds, where a tolerance is the allowed gap between a detected and a true start. Many of these tolerances are longer than fine-grained actions, which average 1.7 s in Assembly101 [221], 2.0 s in HD-EPIC [200], and 3.1 s in EPIC-KITCHENS-100 [222]. Our calculation from the released annotations gives medians of 1.18 s and 1.62 s for the latter two datasets,

both under two seconds. A tolerance of several seconds can therefore credit a prediction that arrives after the action ends. Boundary accuracy thus needs to be interpreted alongside detection delay.

3.3.5 What to Keep: Compression, Representation, and Redaction

Aligned streams and structured records. We compare representations by the structure they retain: word–frame timing in an aligned stream and recurring relationships in event records. LiveCC [223] interleaves video frames and transcript words at their timestamps, preserving temporal alignment during training. Imprint [193] extracts person–action–object records with time intervals, then consolidates them using recurrence, recency, and distinctiveness. Hierarchical caption summaries instead describe a video at several time scales [224]. Compared with those of EgoRAG [94], Imprint's records improve long-horizon question answering and reduce storage and retrieval cost.

Sensitive-content filtering. Filtering before transmission or storage can reduce what is disclosed and limit what the retained evidence can answer. Kirigami [140] filters speech locally and keeps the remaining audio for activity recognition. It measures both effects: how much speech can still be recovered and how accurately the remaining audio identifies activities. The appropriate trade-off depends on the task: detecting whether the user is free can use non-speech cues, while retaining a spoken commitment requires its words.

3.3.6 Whether Enough Has Been Observed: Interpretation and Readiness

Evidence sufficiency and response timing. Readiness decides whether the evidence at hand is enough for the next decision. A query can arrive before its answer is observable. For such queries, StreamReady [194] learns a readiness score from evolving visual context and observes until the score passes the answering threshold. Sufficient evidence still leaves the action open: even when Sensing has located the landmark the user is looking for, Action can still hold back spoken directions while the user is talking to someone.

Fresh input and event confirmation. Processing an observation quickly does not confirm that the event needed for the next decision has occurred. Robot controllers make the distinction explicit when observed contact events confirm transitions between task phases [141]. An always-on assistant faces the same distinction when a delivery notice arrives before the parcel has actually reached the user.

Answers from the user. When evidence is missing, the assistant can ask, and the answer becomes a new observation. Systems differ in what the answer updates. PrISM [135] tracks procedural steps from a smartwatch, asks about the user's current step when its estimate is uncertain, and uses the answer to refine that estimate in real time. The Egocentric Co-Pilot [151] chooses between answering directly and asking a short clarification question, then adds the reply to its context before acting. SpeechLess [134] pairs each low-confidence interpretation with a yes/no confirmation, so the user's answer decides what enters memory. EgoAsk [153] asks questions during daily activity to fill gaps in a household robot's knowledge of the user's objects. Users in its study found it less effort to keep track of what the robot still needed to learn when the robot asked than when they had to teach it themselves.

3.3.7 Where the Computation Runs: On-Device, Edge, and Cloud

Local processing and computation splits. Computation can run on the device, as EgoCHARM's [113] activity recognition does, or be split with a paired phone or the cloud. Split inference instead assigns part of the workload elsewhere, selecting the split from computation costs, the amount of data passed between the two parts, and network conditions [225]. DeepWear [184] partitions inference in this way between a wearable and its paired phone over Bluetooth, balancing latency and energy without Internet access. For a whole assistant, OpenJarvis [226] lets cloud models propose edits to a personal stack during search and keeps only edits that do not worsen performance. It then runs the result entirely on the device, where it matches cloud accuracy on several benchmarks at a small fraction of the cost per query.

Content that crosses the link. Placement also determines which representation leaves the device. Shredder [195] adds learned noise to the intermediate features sent for cloud inference and compares information

leakage with inference accuracy. Choosing a split point and protecting the transmitted representation are separate decisions: keeping computation local avoids that transfer, while feature protection changes the information available remotely. A useful split returns task-relevant evidence within the assistant's latency, energy, and privacy constraints.

3.3.8 Where the Result Goes: Routing

Different recipients require different evidence. Dispider [144], for example, supplies compressed clip features to a lightweight decision process and retrieves historical clips for a separate response module, continuing to monitor during response generation. MIRIX [20] routes to memory instead: it combines incoming content with retrieved context, and a central memory manager then selects specialized managers for updates. Dispider separates evidence for immediate control and response generation, while MIRIX separates choosing a persistent destination from maintaining its records.

Evidence can also go to a background process, whose result can arrive after the conversation has changed. Thinking Machines' interaction models [196] keep receiving the user's input while they respond, and they pass conversational context to asynchronous reasoning that runs in the background. MoshiRAG [227] applies asynchronous retrieval to external knowledge. Retrieved text enters ongoing speech generation while the model continues the conversation. A returned result can therefore be out of date, for example when the user has cancelled the request in the meantime, so the assistant needs to check for such changes before using it. Routing chooses both the receiving process and the context sent with the evidence. For example, a brief cue that the user is busy serves only the current decision, while a new commitment needs a record with source and timing information for later correction.

3.3.9 Learning the Sensing Policy

Beyond any single decision, the sensing policy itself can be learned. Most learning in the reviewed work happens before deployment. For example, SVTAS-RL [209] optimizes online segmentation with reinforcement learning, and VAGEN [228] rewards a multi-turn agent for describing the current visual state and predicting the next one. Other works learn readiness and triggering: StreamReady [194] learns when streaming evidence suffices to answer, and StreamMind [125] learns a gate that decides when its language model runs. EyeRobot [229] and EyeRobot 2.0 [155] learn where to look by rewarding views that help their action policy reproduce the demonstrated actions. These policies stay fixed once the assistant runs. A self-evolving loop would instead revise its triggers, windows, or checks from the outcomes of its own deployment, for example lowering a threshold after a missed event that Action needed. We found no reported evidence in the reviewed systems of durable sensing-policy revision from deployment outcomes and subsequent use of the revised policy.

Across the eight decisions, the reviewed methods share three gaps, concerning evaluation, what is kept, and placement. Most are evaluated against known queries or labels, which does not show whether observation between requests kept the events a later decision needed. Activity categories are often fixed before deployment, and each stored representation preserves only the information selected when it is created. Details needed to interpret later routines or answer future questions can therefore be lost. Placement is chosen by accuracy, latency, energy, or cost, and the reviewed methods protect what leaves the device in a separate step, without weighing it in that choice. These gaps point to three directions: judging sensing by its coverage of later needs, guiding observation with retained context through $\mathrm{M} \rightarrow \mathrm{S}$, and weighing what information leaves the device when deciding where computation runs.

3.4 Sensing Coupling Across the Loop

The local sensing policy determines how observations are acquired, interpreted, and delivered. In a continuing loop, the policy's decisions also depend on what the assistant remembers and aims to do. We examine what the two incoming channels, $\mathrm{M} \rightarrow \mathrm{S}$ and $\mathrm{A} \rightarrow \mathrm{S}$, supply to Sensing, which decisions they guide, and why that guidance helps or fails. Table 7 compares representative implementations by their guidance, sensing use, and reported evaluation.

3.4.1 Local Decisions and Coupling Boundaries

Sensing's local policy decides what to capture and how to interpret it, and these decisions can also be guided by what crosses its coupling boundaries from Memory ($M \to S$) and from Action ($A \to S$). We classify the guidance for these decisions by its source, which is Sensing itself, Memory, or Action, as Table 6 illustrates. Local processing in Sensing acquires and interprets observations, for example by triggering capture from motion in the current stream or by using a short frame buffer to interpret the scene. $M \to S$ is involved when retained personal state guides acquisition or interpretation, such as using a persistent identity record to recognize a person. $A \to S$ is involved when an intended action, plan, or execution state supplies an information need or observation target. All three classes of sensing can receive this guidance.

These channels supply information, but the local sensing policy makes the choice, weighing that information against current inputs and the shared conditions. The two channels can inform the same choice. In the running example, a remembered routine suggests when to check whether the user is free, while Action's plan to remind specifies what the check must confirm. Memory can also reach Sensing indirectly through Action: personal context first shapes the question the assistant asks, and that question then sets the next observation. Over time, two kinds of change can follow. Feedback can correct the stored routine, which changes later checks without changing how they are chosen. Self-evolution instead revises the mechanism itself, such as how Memory selects guidance, how much weight Sensing gives an old routine, or how Action turns a plan into an observation request.

Table 6 Local Sensing decisions and incoming guidance.

Relation	Description	Example
Local policy	Makes acquisition and interpretation choices	Current motion triggers image capture
$M \to S$	Retained personal state guides Sensing	A stored identity helps recognize a person
$A \to S$	Action context supplies observation targets	Action's plan to remind prompts a check of whether the user is free

3.4.2 Memory to Sensing: the $M \to S$ Channel

Choosing further observations with retained context. Retained context can guide which evidence the assistant acquires next: the agent in IPIBench and Em-Garde direct attention within a stream, while ProAct decides how much new evidence to acquire. The agent in IPIBench [129] stores a user's monitoring request as a monitoring task and turns it into text descriptions of the awaited event. Each new stretch of video is compared with these descriptions before the agent responds. An ablation test that disables this comparison lowers the scores across the monitoring tasks. Em-Garde [128] similarly turns a pending query into short descriptions of the awaited event, once when the query arrives, and compares every new stretch of video with them. ProAct [152], by contrast, checks how much of a predicted need its memory already covers before it searches. High coverage reuses stored evidence, partial coverage limits the search to missing subtopics, and low coverage starts an iterative search. ProAct's evaluation compares directed with undirected idle-time search and does not isolate the coverage check. If the stored task or the coverage record is out of date, the assistant can watch for the wrong event or skip a search it still needs.

Interpreting new observations with retained context. Once observations arrive, retained context can help resolve their ambiguity, both about who is present and about what a brief request means. M3-Agent [95] stores an identity node for each known person and matches faces and voices in incoming clips to these nodes before producing new event descriptions. It also accumulates evidence for face–voice equivalences, so repeated consistent associations outweigh occasional conflicting links, and removing these links lowers later question-answering accuracy in its ablation. Consistent identities come with a risk, since a wrong match can propagate to later records. SpeechLess [134] uses personal spatial episodes to infer missing parts of a short utterance or context-only request, which spares the user repeated explanation. In its interaction study, articulation effort falls, but inference delay rises and accuracy drops for some abbreviated modes. Its risk is different: an overly specific interpretation can make an ambiguous request appear settled before the user has confirmed it.

Identity guidance can also operate before language interpretation. Personal VAD [112] detects the target speaker with an enrolled voice embedding, a numeric summary of the user's voice stored on the device. VoiceFilter-Lite [230] uses the same kind of personal reference to suppress competing speech before recognition. The former controls which speech activates processing, while the latter controls what remains in the signal. Their evaluations test detection and recognition, respectively: two points at which another person's speech can enter the assistant's evidence. Similarly, Memento [231] applies retained references to visual localization: an object named in a retrieved memory becomes a detector target in the current view. These methods connect persistent identity or object references to current evidence without requiring the user to name them again.

Why memory-guided sensing is difficult. The difficulty of memory guidance lies in whether the retained state still reflects the user's preferences and current situation. Both can change after they are recorded, while the assistant keeps trusting the old record, which leads to two failures. First, **the assistant can misread new evidence**. The observations still enter the record, so a later check can find the mistake. Second, **the assistant can stop looking where the change happened**. A missed observation leaves nothing to check, as when a prediction-error trigger cannot notice a change in a stream it no longer samples. Asking fails the same way, since the user's reply is also an observation. In PAHF [232], simulated users change their preferences. An agent that asks only before acting stops asking once its notes look certain, even after they have become outdated, and adapts poorly to the new preferences. Feedback after acting avoids the problem, since agents that receive it can correct those notes without having to ask.

Correction therefore depends on chances to revisit what the assistant believes. STALE [233] tests whether agents notice that a later observation has implicitly invalidated a stored belief, which is the cue a fresh check needs. Revisiting can cost movement, renewed access, or the user's help, so an assistant can rely on a remembered state for routine decisions, look further when it is uncertain, and still run occasional checks that would reveal a changed routine, identity, or permission. Without such checks, the retained state that guides observation can keep the assistant from seeing the very evidence that would correct it, especially when the assistant observes only selectively.

3.4.3 Action to Sensing: the A→S Channel

Action guides Sensing at three points: before, during, and after execution. What an action intends and expects sets what Sensing looks at next.

Reading what a pending action needs. Before an action, Sensing obtains the evidence that the pending decision still lacks. General planners identify missing requirements and ask for them before planning [234], and calibrated robot planners ask for help when several actions remain plausible [235]. Personal assistants acquire such evidence as part of helping the user. ContextAgent [96], for example, reads location and then weather when a planned suggestion depends on both, and Alpha-Service [131] sets visual triggers and queries for the service tasks it dispatches. PARE [236] lets a proactive assistant run read-only queries across apps while a simulated user acts, before it asks permission to execute. On smart glasses, the Egocentric Co-Pilot [151] orchestrates perception and web tools for questions about the wearer's current situation. Such a question or read helps only when it addresses the uncertainty of the pending decision and its result returns in time.

Monitoring requirements during execution. Some requirements apply throughout an action. General robot monitors translate task constraints into executable visual checks [150], and such a check is useful only if its input contains the signal needed to detect the failure. In manipulation, the action itself can remove such a signal, for example when a grasped object blocks a wrist camera. Vision in Action [156] learns from human demonstrations to move a head camera on a robot neck and look past shelves and bags that block the wrist cameras. EyeRobot 2.0 [155] instead uses a single stereo camera without wrist cameras: its two eye viewpoints swivel to fixate the object the task needs next, and it processes the region around that point at higher resolution. When grasped objects block the wrist cameras, EyeRobot 2.0 succeeds about twice as often as the common wrist-camera setup. For an ongoing task, the method in Pro^{2}Bench [237] lets the planner supply visual cues for step progress and completion and uses the times of plan updates to choose which video clips to check. SentinelBench [238] tests monitoring of live web applications while an agent waits

for a condition. The same monitoring problem arises when a personal assistant checks whether a background task still satisfies the user's instructions.

Checking effects after execution. By checking an effect we mean reading the state that an action produced, which differs from receiving a confirmation that the action was issued. General graphical-user-interface (GUI) agents record an expected effect and verify it before continuing [154]. Systems that act for a user check effects at several depths, from comparing results with expectations to changing what they keep for later. AssistantX [2], deployed for a month and a half in a real office, compares message replies, robot localization, and its locker's lock signal with the expected outcome of each step, and ABot-Claw [6] uses a critic to evaluate progress online and trigger local correction or replanning. Mobile-Agent-E [136] keeps the lessons from its before-and-after screen comparisons for later tasks. KnowAct-GUIClaw [239] goes further and uses screen evidence from a failed step to repair the stored skill in place. Other systems use the next state as a training signal: OpenClaw-RL [240] learns from the user reply, tool output, or interface change that follows each action. The user's behavior can also be the observed effect. GAP [241] estimates where a home robot should place objects from whether the simulated user builds on its placement or undoes it. Products document similar checks: OpenAI's Dot [16] checks the results of tasks it creates, and Grok Bot [242] confirms that requested visual changes were applied. LIMBO [243] tests late commits and duplicate effects within bounded tool tasks, while late effects across interactions and retained commitments remain untested.

Why action-guided sensing is difficult. First, **a check has to come at the right time**. The effects of an action can be delayed: some show up at once, such as a tool confirmation or the user's reply, while others take hours, days, or weeks, and an observation stays valid only for a while. A check that comes too early finds no result and can trigger a needless retry. A check that comes too late leaves the actions taken before it to build on an unconfirmed assumption, and by then other changes can hide what the action did. Even a missing result is ambiguous, since it can reflect failure, delay, or a view that hides the effect, and a cancelled request makes its pending checks obsolete. Second, **the source often sees only part of the resulting state**, a limit known as partial observability. A tool receipt can confirm only that a command was accepted, while visible completion or a side effect requires inspecting the resulting state. One response to this limit is to act: interactive perception [244, 245] studies actions that reveal otherwise inaccessible properties, which makes acquisition part of the task. In personal assistance, such revealing actions have costs: asking the user to look again adds effort, moving a robot adds physical cost, and opening another page requires access. Useful A→S guidance therefore keeps checks tied to the active task and schedules them for when the expected effects become observable.

3.5 Evaluation

3.5.1 Evaluation Metrics

An assistant cannot act on an event it never captured, so sensing evaluation has to test coverage as well as recognition. Table 8 lists the benchmarks and evaluation protocols for sensing, grouped by the operation they test. Their metrics answer four questions. **Coverage** asks whether the events that matter were captured. Event recall measures the share of reference events the system caught, and the false-trigger rate measures how often it activates without need. Measuring what was missed needs an event reference that does not come from the observations the system selected. For on-demand sensing, coverage also includes whether the assistant checks retained state again before relying on it. **Interpretation** asks how well the system understands what it captured, measured by recognition accuracy and segment-level scores. **Timeliness** asks when enough evidence becomes available and when a response follows. A latency comparison needs explicit start and end points, including whether capture, processing, and transmission are counted. **Cost** covers sensor activity, computation, and communication, and for on-demand sensing the cost of each check. Reporting the four together shows the trade-offs of keeping observation going between requests.

3.5.2 Benchmarks and Evaluation Settings

Perception baselines. Perception baselines evaluate interpretation of supplied input, leaving acquisition decisions untested [249, 250, 251, 252]. WearVQA [253], for example, targets smart-glasses use: questions ask

Table 7 **Representative implementations of M→S and A→S.**

Work	Guidance supplied	Use in Sensing	Reported evaluation	URL
M→S: retained state guides observation and interpretation				
ProAgent [130]	Retained personas	Interpret the current view	Persona ablation	/–
ProAct [152]	Memory coverage	Search only missing evidence	Directed-search comparison	/
Personal VAD [112]	Enrolled voice embedding	Detect target-speaker speech	Detection precision–recall	/–
VoiceFilter-Lite [230]	Enrolled voice embedding	Suppress competing speech	Recognition error/noise	/–
SocialMind [246]	Stored face identities	Identify known partners	Persona-module ablation	/–
M3-Agent [95]	Face/voice identity nodes	Ground new event records	Identity-link ablation	/
Memento [231]	Stored memory referents	Locate objects in view	Multi-day usage study	/–
AiGet [247]	User interests/expertise	Interpret scene familiarity	Profile/rule ablations	/
SpeechLess [134]	Personal spatial episodes	Resolve partial intent	Speech-mode comparison	/–
IPIBench [129]	Retained monitoring tasks	Match task cues to frames	Control/gating ablations	/
A→S: action context supplies information needs and observation targets				
ContextAgent [96]	Planned tool dependencies	Read location, then weather	Tool-selection accuracy	/
Alpha-Service [131]	Dispatched service tasks	Set visual triggers/queries	Service demonstrations	/–
PARE [236]	Pending proactive action	Read-only queries across apps	Simulated active users	/
Pro^{2}Bench [237]	Step cues and timestamps	Select clips and track steps	Plan-condition comparison	/–
AssistantX [2]	Expected task outcomes	Check messages/robot state	Agent-module ablations	/
Mobile-Agent-E [136]	Executed phone action	Compare before/after screens	Complex mobile tasks	/
GAP [241]	Robot object placement	Observe whether it is undone	Multi-day simulated homes	/–
PITCH [248]	Earlier daily plan	Interpret later completion reports	Daily-dialogue analysis	/–

models to identify visible objects or check an assembly step against a manual, including images affected by blur, occlusion, or poor lighting. ScreenSpot-Pro [254] targets professional desktop software instead, such as CAD and image-editing tools: given a high-resolution screenshot and an instruction, the model locates the intended interface element. Such baselines also show what current video models still do poorly. On TOMATO [255], which asks about direction, counting, and the order of actions in short clips, the best model scores 37.9% against 95.2% for people. In online video, OVO-Bench [256] puts the best model about 17 points below people at identifying a sequence of actions, with streaming models further behind, and on fine-grained kitchen actions in HD-EPIC [200] the best model scores 43.0% against 92.5%.

Capture coverage and cost. Another group of evaluations starts from real-world recordings and tests whether a system can capture less, at lower cost, without losing performance. EgoTrigger [121] compares continuous capture, uniform frame sampling, and audio-triggered capture, which records for a fixed duration whenever it hears hand–object interaction sounds, on QA-Ego4D and HME-QA. The audio-triggered capture keeps comparable memory question-answering performance with roughly half as many frames as continuous capture. On a Raspberry Pi 5 prototype, EgoTrigger also lowers average power by about a sixth compared with continuous capture.

Organizing supplied streams. Other tests fix the stream and ask whether what was captured is organized well. Within a supplied stream, a matched frame budget makes SimpleStream's [257] recent-frame baseline a useful control for more elaborate history selection. OnlineTAS [210] measures both frame accuracy and event segmentation, the latter with an edit score that penalizes fragmented or out-of-order segments. On 50Salads, its online post-processing raises the edit score from 28.8 to 71.5 while frame accuracy falls from 80.9% to 77.7%. Frame accuracy, event continuity, and capture savings therefore measure different things, and none of them can stand in for another.

Readiness and response timing. Readiness benchmarks score timing together with correctness. Timing matters because no request marks the moment to respond, so judging when enough evidence has arrived is part of the task. For example, in step-by-step guidance on smart glasses, a warning given before the user has picked up the wrong part only confuses them, and one given after the part is fastened comes too late to prevent rework. StreamReady [194] annotates the time windows containing answer evidence and

scores whether a response was premature or delayed. OVO-Bench [256], ESTP-Bench [258], TeleEgo [259], EgoPro-Bench [260], and OmniMMI [261] similarly ask whether a response uses the evidence available so far. WearVox [262] covers the decision not to respond, testing whether an assistant ignores side talk, speech not addressed to it.

Checking again. Few evaluations score the decision to observe again. Retained state can go stale without any signal: a gate change posted on the airline's website leaves the stored itinerary looking correct until the assistant checks again. VibeLifeBench [199] builds multi-week tasks in a simulated world of 22 services whose changes are often silent, so a high score requires inspecting the world again without being told to. MCB [198] instead separates storing, one-time use, re-verification, and asking the user, which shows where a failure occurs.

Guidance from Memory and Action. Coupling evaluations examine how retained context and action plans guide observation. Tests of M$\rightarrow$S guidance include Personal VAD [112] and IPIBench [129]. Personal VAD uses an enrolled voice, and IPIBench tests monitoring requests that users add, edit, or cancel during a stream. PAHF [232] and STALE [233] let retained beliefs go out of date without notice and score the agent's questions and judgments, but no reviewed test counts the observations outdated retained state led the assistant to skip.

For A$\rightarrow$S guidance, response quality alone does not show whether a plan guided observation in time. The method in Pro^{2}Bench [237] handles both monitoring and response. Run with no plan, a generated plan, and the correct plan, it ranks these conditions differently by response quality and by intervention timing. SentinelBench [238] also scores reaction time and waiting cost. In manipulation, TAVIS [263] measures how long before the grasp a policy's gaze arrives at the target. Imitation policies match the lead time of the human demonstrations but move the head in more, smaller steps, which success rate does not reveal.

What Sensing Misses Is Rarely Evaluated

Always-on sensing cannot yet afford high frame rates, and what it misses is lost. Power limits keep always-on wearables at low frame rates, yet many everyday actions last under a second and can fall between frames. Unlike a misread frame, an uncaptured action leaves no record and no error signal, and the assistant cannot replay the user's day to recover it.

Stale memory hides the evidence that would correct it. When the world changes without notice, an outdated record can keep the assistant from looking where the change happened, and no reviewed test counts the observations it skipped.

No reviewed work shows that a revised sensing policy misses fewer events. Showing this needs an independent record of what happened and a comparison before and after the revision.

Table 8 Sensing benchmarks and evaluation protocols.

Benchmark / protocol	Setting	Evaluated content	Framework	URL
Perception baselines: supplied input				
ExtraSensory [264]	Phone/watch sensors	Daily context recognition	π_S	–/–
Ego4D [265]	Egocentric video	Activity, recall and social understanding	π_S	–/–
WearVQA [253]	Wearable capture	VQA under authentic capture conditions	π_S	–/–
EgoNight [250]	Aligned day/night video	Day/night VQA, retrieval and depth	π_S	–/–
EgoSound [249]	Egocentric audio/video	Sound understanding and audio–visual QA	π_S	–/–
EgoSAT [266]	Egocentric stream	Interaction QA and confidence calibration	π_S, c_t: unc.	–/–
EgoCross [251]	Specialist-domain clips	VQA beyond daily-life domains	π_S	–/–
ScreenSpot-Pro [254]	Screens	GUI grounding on high-resolution screens	π_S	–/–
WebPII [267]	Screenshots	Sensitive-region detection	π_S, c_t: priv.	–/–
GUIGuard-Bench [169]	GUI agent trajectories	Privacy regions, risk, task need	π_S, c_t: priv.	–/–
SmartHome-Bench [268]	Home-camera clips	Anomaly detection with rationales	π_S	–/–
ENACT [252]	Simulated household replays	Forward and inverse world modeling	π_S	–/–
Continuous and triggered capture: coverage and cost				
EgoLife [94]	Multi-day recording	Event recall and reasoning	S $\to$ M, $\mathcal{B}$	–/–
EgoTrigger [121]	Recorded egocentric video	Frame count, QA, and power	S $\to$ M, c_t: res.	–/–
Ego-METAS [187]	Continuous streams	Sensor selection under energy budgets	π_S, c_t: res.	–/–
Organizing supplied streams				
OnlineTAS [210]	Arriving video	Frame and segment scores	π_S	–/–
Kinetics-GEBD [218]	Supplied video	Event boundaries without action classes	π_S	–/–
SimpleStream [257]	Frame budget	Recent frames versus history use	S $\to$ M, c_t: res.	–/–
Flash-VStream [205]	Long video stream	QA on the video so far	π_S	–/–
SensorQA [269]	Phone/watch histories	Activity QA over days to weeks	S $\to$ M, c_t: res.	–/–
HD-EPIC [200]	Multi-day kitchen video	Recipe, object-motion and gaze VQA	S $\to$ M	–/–
SENRI-Bench [270]	Hours-long chest camera	Event–place recall in both directions	S $\to$ M	–/–
MuRAL [271]	Multi-resident sensors	Resident attribution and descriptions	π_S	–/–
Readiness and response timing				
TeleEgo [259]	Egocentric stream	Recall, accuracy and timeliness	S $\to$ A, S $\to$ M	–/–
OVO-Bench [256]	Video prefix	Recall and response under a prefix	S $\to$ A	–/–
StreamReady [194]	Evidence windows	Answer correctness and timing	S $\to$ A	–/–
ESTP-Bench [258]	Evolving questions	Timely responses	S $\to$ A	–/–
EgoPro-Bench [260]	Egocentric stream	Intervention timing and memory consistency	S $\to$ A, M $\to$ A, $\notin \mathcal{R}$	–/–
Full-Duplex-Bench [272]	Spoken dialogue	Pauses, backchannels, turns, interruptions	S $\to$ A	–/–
WearVox [262]	Wearable audio	Side-talk rejection	S $\to$ A, $\notin \mathcal{R}$	–/–
ProactiveVideoQA [273]	Four video sources	Reply timing (PAUC)	S $\to$ A	–/–
HomeSafe-Bench [274]	Robot hazard clips	Alert timing before impact	S $\to$ A, $\notin \mathcal{R}$, c_t: res.	–/–
On-demand: checking again				
VibeLifeBench [199]	Silently changing world	Re-inspection over weeks	M $\to$ S, $\notin \mathcal{R}$, $\mathcal{B}$	–/–
MCB [198]	Memory decisions	Verification versus clarification	A $\to$ S, M $\to$ S, c_t: unc.	–/–
EnvTrustBench [167]	Misleading tool outputs	Verification before acting on claims	A $\to$ S, c_t: unc.	–/–
PInVerify [275]	Multi-view home scans	Verify the item before acting	π_S, c_t: res.	–/–
EXPRESS-Bench [276]	Simulated home scans	Grounded answers, path cost	π_S, c_t: res.	–/–
LA-EQA [277]	Scenes across days	Recall or re-explore, path cost	M $\to$ S, $\mathcal{B}$, c_t: res.	–/–
Guidance from Memory and Action				
Personal VAD [112]	M $\to$ S: enrolled speaker	Target-speaker detection	M $\to$ S, c_t: res.	–/–
VoiceFilter-Lite [230]	M $\to$ S: enrolled speaker	Recognition with competing speakers	M $\to$ S, c_t: res.	–/–
IPIBench [129]	M $\to$ S: stored tasks	Monitoring tasks added, edited or cancelled	S $\to$ A, M $\to$ A, M $\to$ S	–/–
Pro^{2}Bench [237]	A $\to$ S: step cues	Intervention timing across plan conditions	S $\to$ A, A $\to$ S	–/–
SentinelBench [238]	A $\to$ S: waiting targets	Condition-based waiting: success, delay, cost	S $\to$ A, A $\to$ S, c_t: res.	–/–
TAVIS [263]	A $\to$ S: grasp targets	Gaze lead time, head versus fixed camera	A $\to$ S	–/–
VABench [278]	Tabletop manipulation	Active versus fixed camera views	A $\to$ S, S $\to$ A	–/–
TVRBench [279]	Simulated indoor homes	Reach a target view actively	A $\to$ S, S $\to$ A	–/–

Framework: π_S marks a protocol that scores the Sensing policy alone, with no coupling channel; $X \to Y$ denotes a coupling channel; c_t gives the shared conditions that are scored; $\notin \mathcal{R}$ marks a scored decision at a step outside $\mathcal{R}$, where no explicit request is being carried out; $\mathcal{B}$ marks state that is retained when an interaction ends and is used later. A channel is marked only when the score of the protocol depends on that flow between modules. No protocol in this table scores a mechanism revision (θ_t). Condition abbreviations: **priv.** privacy, **res.** resources, **unc.** uncertainty.

4 Memory

Memory keeps what the assistant learns about the user and hands it back when a decision needs it or a commitment falls due. Its records come both from what the user says and from what the assistant infers, such as habits, and they build up over months or years. Over such a span, the main difficulty is that a correct record can become wrong when the user's preferences or circumstances change, and a wrong record keeps shaping later assistance until something corrects it. Memory has two further duties: it holds commitments until their moment arrives, and it records who supplied each item and who is allowed to use it. For the continuing loop, Memory provides the *state durability* of §2.1: it keeps whatever must outlast an interaction, such as task progress or a pending commitment, so that the assistant can resume where it stopped. For a self-evolving loop, it also keeps the assistant's own actions and their outcomes, the record from which that loop revises the mechanism, including how Memory admits, uses, and retires information. The two roles draw on records of different times: resuming assistance needs mainly what holds now and what is still pending, and revision needs what happened. As shown in Figure 8, we therefore describe retrospective, profile, and prospective memory, then cover shared conditions, the local policy, coupling, and evaluation.

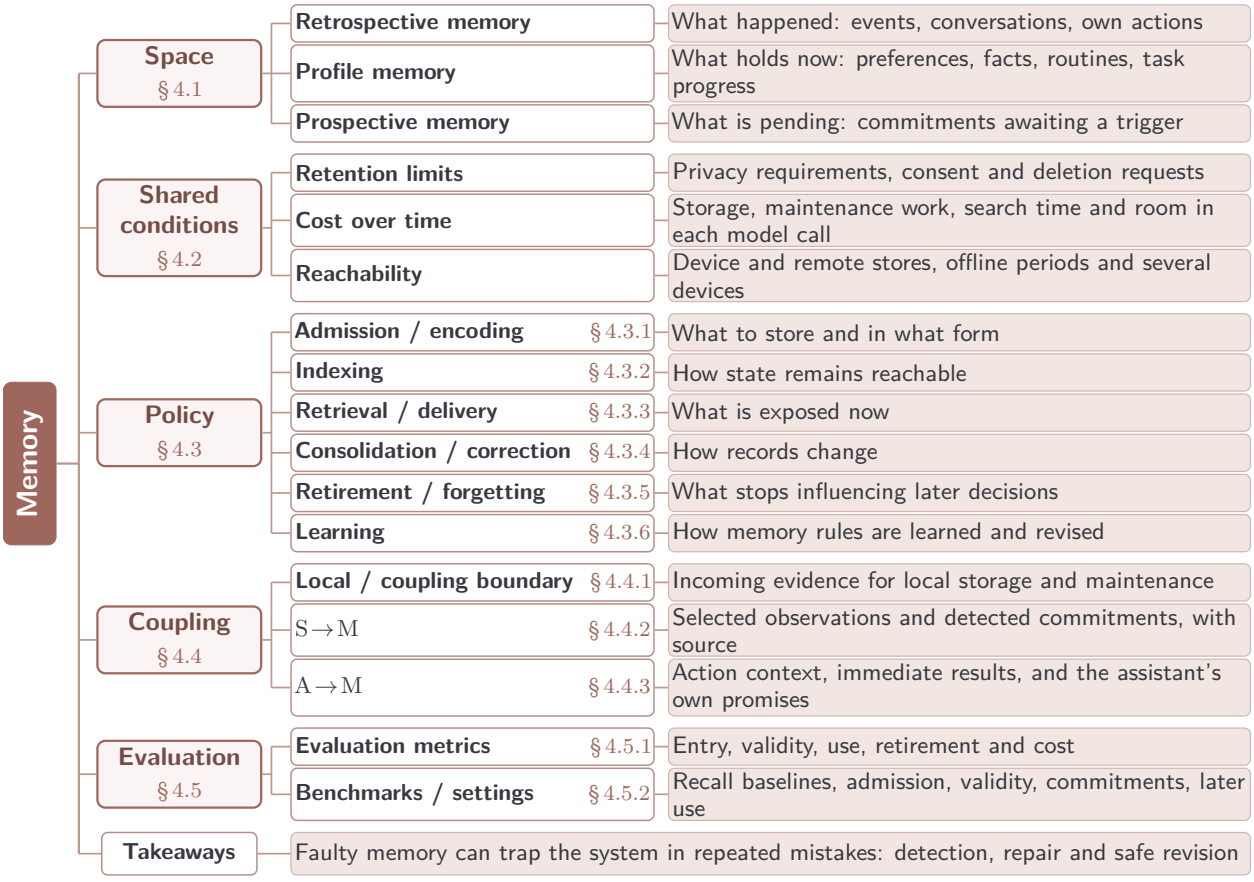


Figure 8 **Memory section structure and taxonomy.**

4.1 Memory Space

The memory space covers what the assistant keeps for later decisions and the format in which it keeps it. §4.1.1 classifies records into **retrospective**, **profile**, and **prospective** memory by the time they describe, and §4.1.2 describes how they are represented and stored. These classes need different maintenance: a preference and a pending task can both outlive the same conversation, yet they are kept current in different ways. A long context does not remove this need. It makes more history available within one model call, but how well a model uses that history still depends on where the information sits and on the task [280, 281]. Because

retained records serve later decisions as the user's circumstances change, a record's value depends on the decisions it supports and on the work needed to keep it valid.

4.1.1 What Persists: Past, Present, and Future

Retrospective memory: what happened. Table 9 lists the different classes of memory. Retrospective memory keeps records of observed events, conversations, and the assistant's own actions together with the responses they drew. Each record keeps its source and time, so a later decision can reinterpret it when new evidence arrives. Its typical failure is evidence that is lost or no longer shows where it came from. MemMachine [285] and ABot-AgentOS [290] guard against this by keeping provenance, a trace of where each record came from. With this trace, a record can be revisited. The records come from four sources: what the user says, what the assistant observes, what it infers, and what its actions leave behind. Conversational memory systems cover the first and third: they retain dialogue histories and derive higher-level descriptions from them [18]. MemGPT/Letta [19] and MemoryOS [284] then decide which parts of that history enter a bounded model context, the limited input a model reads in one call. Observation extends the history beyond dialogue. Vinci2 [283] assists over continuous egocentric video, filmed from the wearer's point of view, and recovers visual detail from compressed records, and Imprint [193] compresses egocentric streams online for later questions. Windows Recall [139] keeps a searchable screen history on the user's computer. The assistant's own actions complete this history: AssistantX [2] keeps its executed tasks and their checked results until an instruction is complete, then clears them, and KnowAct-GUIClaw [239] records the results of subtasks. Keeping such records across interactions provides *state durability*, and the actions with their outcomes are also the trajectory a self-evolving loop learns from. As records accumulate, the challenge is to find the ones that matter for the current decision.

Profile memory: what holds now. Profile memory keeps what holds now about the user, their world, and their ongoing tasks. It covers preferences, facts, relationships, and routines, together with current goals, plans, and progress. Keeping these items valid is the main maintenance problem [233, 321], since a stale value can stay in use after it stops being true. Each item has a validity that can be standing, temporary, or expired. Temporal and relational stores maintain such items across sessions by recording when each item holds and how items connect [294, 92, 296, 297]. Profile content can also come from other users, as when PGraphRAG [300] fills sparse user histories with information from related users. A preference inferred this way needs a source label that keeps it apart from one the user stated. PersonaVLM [301] adds a further source, extending the profile to visual observations. User as Code [302] changes how the profile is represented: it keeps typed personal fields, each holding a declared kind of value, so queries can read them exactly. Ongoing tasks belong to the profile as well: KnowAct-GUIClaw [239] checkpoints unfinished work so that it can be resumed, and HealthClaw [303] maintains longitudinal health state that guides later assistance. Task progress records what has been done, while an obligation to continue at a later moment is a prospective commitment. Carrying current profile content across interactions is *state durability* and lets the assistant fit the user better over time. The profile states what holds now, but some of what the user needs is due at a later moment.

Prospective memory: what is pending. Prospective memory keeps commitments that wait for a time, an event, or a condition, so that the assistant can act at the right moment without a new request. It fails when a reminder is missed, arrives when nothing is due, or repeats. The simplest come from explicit requests, such as a reminder in Pally [312] tied to arriving at or leaving a place. Many other commitments are never stated as requests to the assistant but arise in places such as the user's email, chats, and spoken conversations. Email assistants track conversations that still need the user's reply and replies the user is still waiting for [314]. Texting assistants turn agreements in a chat thread into tracked tasks [312], and wearables turn spoken action items into tasks or drafts [138, 310, 322, 323].[1] The assistant's own promises are commitments too, and the Long-term Task-oriented Agent (Long-term TOA) [315] writes its trigger conditions from the dialogue. Whatever its source, a commitment record holds three fields: a condition, an intended action, and a status. The Prospective Intention Store (PIS) [316] keeps each intention in these fields, with pending, done, or cancelled as the status. User as Code [302], by contrast, writes alert conditions as executable rules

[1] Omi help center: https://help.omi.me/en/articles/13135362-first-steps; developer documentation: https://docs.omi.me/doc/developer/apps/Notifications

Table 9 **Representative memory systems by what they keep.**

Work	Source	Key feature	Input	URL
Retrospective memory				
VoiceMem [282]	Dialogue	Retrieval starts from partial speech	Streaming speech	/–
Vinci2 [283]	Observation	Detail recovered from summaries	Egocentric video	/
Imprint [193]	Observation	Online stream compression	Egocentric streams	/–
LightMem-Ego [137]	Observation	Background capture, later recall	Phones and glasses	/
MemoryBank [18]	Dialogue	Summaries of dialogue history	Conversations	/
MemGPT/Letta [19]	Dialogue	History paged into context	Conversations	/
MemoryOS [284]	Dialogue	Selects history for context	Conversations	/
MemMachine [285]	Dialogue	Keeps where each record came from	Conversations	/
LightMem [286]	Dialogue	Topic groups, deferred upkeep	Conversations	/
Sleep-time compute [287]	Dialogue	Processes context between requests	Conversations	/
Windows Recall [139]	Observation	Searchable screen history	Desktop screen	/–
PANDO [288]	Own actions	Rules and routines after each task	Browser tasks	/–
AWM [289]	Own actions	Workflows induced from past tasks	Web navigation	/
ABot-AgentOS [290]	Observation	Source-linked graphs over time	Robot sensors	/–
STRANDS [4]	Observation	Learns site patterns over months	Mobile robots	/
H-EMV [291]	Observation	Summary tree of robot history	Humanoid robot logs	/
H^2-EMV [292]	Observation	Learns what to forget from feedback	Humanoid robot	/–
AssistantX [2]	Own actions	Tasks with checked results	Office robot logs	/
PhysMem [293]	Own actions	Keeps tested physical hypotheses	Robot interaction	/
Profile memory				
SocialMind [246]	Dialogue	Partner personas, conflicts replaced	AR glasses	/–
Omi Memories [138]	Dialogue	Facts about people and preferences	Wearable audio	/–
AiGet [247]	Personal records	Interest profile steers both prompts	Smart glasses	/
Personal VAD [112]	Personal records	Enrolled voiceprint on device	Audio + voiceprint	/–
Zep [294]	Dialogue	Facts with validity times	Conversations	/
A-MEM [92]	Dialogue	Linked notes that evolve	Agent interactions	/
Mem0 [295]	Dialogue	Add, update, or delete facts	Conversations	/
MOSAIC [296]	Dialogue	Entity records, conflict handling	Conversations	/–
RaMem [297]	Dialogue	Facts anchored to event time	Conversations	/
ChatGPT Memory [298]	Dialogue	Background synthesis over time	Conversations	/–
AdaMem [299]	Dialogue	Learns what to write per user	Conversations	/
PGraphRAG [300]	Related users	Fills sparse user histories	Other users' data	/
PersonaVLM [301]	Observation	Profile from visual input	Images + dialogue	/
User as Code [302]	Personal records	Typed personal fields	Personal data	/
KnowAct-GUIClaw [239]	Own actions	Checkpoints unfinished work	GUI execution	/
HealthClaw [303]	Health records	Longitudinal health state	Health data	/
Noa [304]	Dialogue	Resident profile and dated events	Care-home robot	/–
PHILIA [5]	Observation	People and preferences across robots	Several robots	/–
EgoMem [305]	Observation	Knows users by face and voice	Camera + microphone	/–
MoveCare [158]	Observation	Compares readings with earlier ones	Robot + smart objects	/–
TidyBot [306]	Observation	Placement rules from examples	Example placements	/
Ayub et al. [307]	Observation	Old object contexts fade over time	User-taught images	/–
DynaMem [308]	Observation	Removes moved objects from the map	Mobile robot	/
DovSG [309]	Observation	Local scene-graph updates	Mobile manipulator	/
Prospective memory				
Bee tasks [310]	Speech	Spoken action items to tasks	Wearable audio	/–
Omi tasks [138]	Speech	Action items from recordings	Wearable audio	/–
Memento [231]	Observation	Resurfaces interests in context	AR headset	/–
CareCall [311]	Speech	Closes items on user report	Phone calls	/–
Pally reminders [312]	Request and chat	Place reminders, chat tasks	Messages + location	/–
Orchid [313]	Request and chat	Reminds only if condition holds	iMessage	/–
Inbox Zero [314]	Email	Tracks pending replies	Email	/
Long-term TOA [315]	Own promises	Triggers written from dialogue	Conversations	/–
PIS [316]	Narrative text	Condition, action, and status	Text	/–
GapChat [317]	Dialogue	Event progress from time gaps	Conversations	/
Family reminders [318]	Request and chat	Reminds when time and scene match	Home robot camera	/–
ElliQ reminders [319]	Request and chat	Appointments told by the user	Social robot	/–
PALRO [320]	Observation	Reminds when daily vitals are missing	Robot + vital signs	/–

over the typed user state, which makes them part of the mechanism (§2.4). PM-Bench [324] tests whether such commitments resurface when due, and TriggerBench [325] tests whether constraints stated without a request resurface when a trigger arrives. The research priority is reliable fulfillment of still-valid commitments: choosing and adapting when and how to check their triggers as user needs and privacy requirements change, within power and latency budgets.

4.1.2 Representations and Storage

Across the three classes, the format in which a record is stored decides which details a later decision can recover and how easily a wrong record can be corrected or deleted.

Raw records and summaries. Raw records, such as full transcripts, logs, or screen histories, keep the original content, including its most sensitive details. A long raw history can exceed the model's context, so systems move parts of it in and out or search it on demand [19, 326]. Summaries are compact and easier to read [18, 286], but they drop details and uncertainty that some later questions need.

Structured records and embeddings. Structured formats make relations among people, entities, and times explicit [294, 92], but a fact spread across nodes and links is hard to correct in full. Embeddings, numeric vectors that place similar content close together, support fast similarity search [295]. Similarity does not show which record is current, so Mem0 [295] adds an update step that replaces or deletes an outdated entry when a correction arrives.

Code and dense state. Code, such as a skill library or typed rules [327, 302], and dense numeric state updated by a learned rule [328] store procedures and past interaction compactly. Because code specifies how a policy operates, it belongs to the mechanism, and changing it is a revision.

Parametric memory. Parametric memory blurs the line between stored records and the mechanism, because it stores content in model parameters. MemoryLLM [329] adds memory parameters to each layer of the network and overwrites a small part of them with each new text, and Cartridges [161] trains a small key–value cache for each document collection, matching the quality of placing the whole collection in the prompt with 38.6 times less memory. Such memory is compact, but a single record in it is hard to inspect, correct, or delete.

The choice of a storage architecture. Storage design belongs to the implementation of §2.4. Here we ask how it changes what the assistant can keep and read back. An assistant that runs all day takes in a steady stream of conversations, screens, and sensor readings, as multi-day egocentric recordings show [94]. Keeping more of it costs storage, and writing, indexing, and searching it costs computation and energy, which are scarce on wearables and phones. Storage design therefore decides how much history the assistant can afford to keep. Memory operating systems, for example, organize storage into hierarchical tiers [330]. Retrieving more history also does not always help. One comparison finds that broad retrieval helps factual question answering but can crowd out the context needed for actions over several steps [331]. Compact learned memory faces a different limit over long use. Metis [328] stores history in a dense memory state, and in its tests recall of earlier personal facts declines as later facts are added.

4.2 Shared Conditions for Memory

We focus on the three shared conditions that bind Memory most: privacy requirements, resource budgets, and availability. All three come from the gap between writing a record and reading it, which can span months and more than one device.

What is the assistant allowed to keep, and for how long? Privacy requirements and retention permissions decide which content is allowed to become a record and how long it can stay. Three parties set these limits. The user grants or withdraws permission and can ask for deletion, the provider's terms fix how long data is held and where it is processed, and other people involved have expectations about being

recorded. Records from one meeting can therefore fall under different limits. Notes that the user dictates during the meeting can be kept, but what colleagues said in that meeting is their information, and the user's consent does not cover it. Current systems enforce these limits mainly at capture and through user controls. Windows Recall [139] stays off until the user turns it on, lets the user filter apps and websites, and deletes screen snapshots on request. Bee [310] never stores audio and lets the user delete transcripts and summaries at any time. Other limits act when stored content is used, as when an audio model refuses queries about incidental bystander speech [174]. Because a record can be read months after it was written, it has to carry what these limits need: its source, owner, permission, and validity. These fields say who supplied the record, whose information it is, who is allowed to use it, and when it stops holding.

What does keeping and searching cost as the store grows? Even content that the assistant is allowed to keep has a cost, because an always-on assistant adds records for months and each record keeps costing after it is written. Each retained record takes storage and maintenance work. Letting records expire saves this work, but once an event's only accessible record expires, retrieval cannot recover it. Reading also grows more expensive, because search slows as the store grows and each model call has limited room for the records it receives. Memory's resource budget therefore covers storage, maintenance work, search time, and room in each model call.

Can the store be reached when it is needed? Beyond cost, a record helps only if it can be read when a decision needs it, which depends on where the record is kept. When the user carries several devices, for example, a reminder set on the phone helps only if the smart glasses that deliver it can read it. A local store keeps personal data on one machine and stays readable offline, but other devices reach it only through that machine. Windows Recall [139] stores and processes its screen snapshots on the user's computer, and OpenClaw [332] and Hermes Agent [333] keep memory files on the machine that runs the agent. A cloud store stays available while the user's devices are off, but it cannot be read while the device is offline, and network delay slows every read. OpenAI's Dot [16] works on its own cloud computer between conversations. MemPrivacy [334] splits records between the two. It keeps sensitive values on the device and sends only placeholders that mark each value's type to cloud-side memory, so restoring a full record needs the device.

4.3 Memory Policy

A memory error can start in an earlier decision about the same record, so we follow a record through its life. Table 10 organizes the local Memory policy π_M around the decisions made over that life. We compare these decisions by what they preserve, what they cost to store and read, and which records later repairs can reach. The decisions also affect one another. Encoding, for example, can remove details that retrieval later needs, and consolidation can create more records that need correction.

Table 10 Local Memory decisions.

Decision	Main choices	Trade-off	Representative works
Admission	Create/refresh/omit	Omission/repeated influence	AdaMem [299], ProEvent [335]
Encoding	Episode/summary/typed record	Detail/read cost	Vinci2 [283], PIS [316]
Indexing	Identity/time/links	Valid scope/stale links	Zep [294], A-MEM [92], RaMem [297]
Retrieval	Search/filter/push when due	Coverage/delay/false alarms	VoiceMem [282], User as Code [302]
Consolidation	Summarize/derive/defer	Compact reads/lost lineage	LightMem [286], DCPM [336]
Correction	Amend/version/defer	Consistency/early overwrite	MemConflict [321], STALE [233]
Retirement	Expire/close/redact/remove	Deletion reach/later utility	Memora [337], PANDO [288]

4.3.1 What to Store and in What Form: Admission and Encoding

Admission. Admission decides whether to create a record from candidate content, refresh an existing record, or leave the store unchanged. Because the same observation can vary in value across people or tasks, AdaMem [299] learns personalized writing decisions, while PerMem-Bench [338] evaluates which content is worth storing for a particular user. Commitments pose their own admission problem, since a casual agreement in a chat can be the only record of a promise. ProEvent [335] tests this problem by asking agents to maintain

a user's timetable from ongoing instant-messaging chats. The Prospective Intention Store [316] finds the stretches of narrative input that state commitments and turns them into typed records. The user also shapes admission: instructions, confirmation, and retention settings give direct control over what survives, and a private utterance can support the current response without becoming a persistent record. Whoever decides, selective writing reduces storage and exposure, but omitting a commitment can discard the only evidence a later reminder needs. Broad writing, by contrast, preserves more evidence but carries uncertain identities or interpretations into later decisions, and even routine chats can unintentionally corrupt the durable state that later assistance relies on [339]. The choice between selective and broad writing depends on the cost of each error and on whether the original evidence remains available for reconsideration.

Encoding. Encoding then decides which details, uncertainty, and source links survive the write. Textual events and reflections drawn from them support flexible recall [18, 340], and MemGPT and MEM1 keep the cost of reading them within a fixed context size [19, 341]. Keeping the original episodes, as MemMachine [285] does, lets a record be reinterpreted after a correction, but the assistant then pays to retain, access, and eventually remove them. ABot-AgentOS [290] keeps the source through links, writing graphs that record where each item came from and hold both current and past state. Vinci2 [283] keeps compact summaries linked to visual reconstruction, so a query can recover detail the summary left out. Typed fields support exact queries instead. In User as Code [302] and the Prospective Intention Store [316], code, not the model, checks alert conditions and tracks whether a commitment is still pending.

4.3.2 How State Remains Reachable: Indexing and Identity

Indexing. Indexing gives each record the keys that later searches use to find it. Finding similar text is not enough: a note that someone is on a low-sugar diet can concern the user or a family member, and it can last only a month. As coding agents have grown more capable, a recent trend is to keep memory as plain text files that the agent reads, searches, and edits with tool calls. OpenClaw [332] writes Markdown notes and daily logs into the agent's workspace, and Hermes Agent [333] keeps two short files, its own notes and a user profile. Infini Memory [342] also organizes memory as topic documents that the model reads through iterative tool calls. The index then consists of file names and the agent's search tools. In a real deployment, however, the assistant adds notes every day for months, and the files keep growing for as long as it runs. Much of what it observes is also multimodal, such as photos, audio, and screen recordings, which a text file holds only as a caption or a link. A collection of free-form notes this large is hard to manage, and its lines are hard to trace back to their sources. Structured indexes instead add explicit keys and links. Zep [294] represents temporal structure, A-MEM [92] links evolving notes, and MOSAIC [296] organizes records by entity and handles conflicts when saving them. RaMem [297] answers such questions of identity and time by anchoring fragments to event time, mention time, session span, and participants. Across several base language models, its retrieval, which checks when each fragment was valid, improves F1, which scores both the correctness and the completeness of results.

Maintenance and updates. An always-on assistant keeps writing while it runs, so its store has to be updated without pausing the assistant. Some systems update at write time. Graphiti, the engine behind Zep [294], keeps a contradicted fact but closes its validity window, so the old value stays as history. Mem0 [295] now only adds facts and decides at retrieval which dated fact applies.[1] File-based agents edit memory during their own work. Hermes Agent [333] caps each memory file, so the agent must shorten or remove stale entries before a new one fits,[2] and OpenClaw [332] saves key context to the daily note before compacting a long conversation.[3] Other updates run in the background between sessions. ChatGPT [298] and Hindsight [343] synthesize memory from many conversations this way, and ChatGPT rewrites a note about an upcoming trip into the past tense once the trip ends.[4] Each structure also brings its own errors. Correcting a fact in text files means finding every note that repeats it, and when a graph merges a new entity into the wrong node, a valid preference can attach to the wrong person. Incremental updates can also leave stale links, which can be detected by rerunning the searches that previously returned the old record.

[1] Mem0 research: https://mem0.ai/research
[2] Hermes Agent memory: https://hermes-agent.nousresearch.com/docs/user-guide/features/memory
[3] OpenClaw memory: https://docs.openclaw.ai/concepts/memory
[4] ChatGPT dreaming: https://openai.com/index/chatgpt-memory-dreaming/

4.3.3 What Is Exposed Now: Retrieval and Delivery

Retrieval. Retrieval selects which retained records reach the decision at hand. General agent memories keep evidence outside the model's context and search it selectively, for example through session memos over a full-record store [326] or retrieval called from within reasoning [344]. For a personal assistant, the time available for search depends on when the next decision comes, during a conversation or between requests. VoiceMem [282] starts retrieval from partial speech, so context is ready when the user finishes speaking. Searching more widely improves coverage but also mixes in evidence of different reliability, as when preferences inferred from related users enter the pool. DynamicMem [345] identifies retrieval as a major error source in long, multi-application histories. Long-video systems that caption first and then reason over the captions, such as LLoVi [346] and VideoTree [347], work like caption-based wearable memory and share its main failure: a detail left out of the captions cannot be retrieved later. With human-written captions instead of a learned captioner, LLoVi's accuracy rises from 55.2% to 66.0%.

Whether a retrieved record is passed on depends on three things: whether it bears on the current decision, whether it still holds, and who will see or hear the output. The last matters even for a record that is true and relevant. Reading the user's correct address aloud in a room with other people, for example, discloses it to them. Similarly, the assistant can use the user's low-sugar diet to rank restaurants, but it should not mention the diet aloud or pass it to a tool that shows the results to others. SteeM [348] lets the user control how much the assistant relies on memory. OP-Bench [349] measures irrelevant personalization, repetition, and sycophancy, a tendency to agree with the user against the evidence. The Compliance Trap [350] traces how conflicting memories propagate.

Delivery. Retrieval answers a query, whereas a pending commitment reaches Action when its trigger holds, even if no one asks. Delivery therefore relies on a check that runs automatically. Checking more often catches more due commitments but also raises more false alarms, and the designs below differ in when they check. For example, the Prospective Intention Store [316] first filters pending intentions, then checks time conditions and queries hidden information sources before deciding which are due. Some systems check when an event occurs. User as Code [302] runs its rules whenever the typed state changes, and Memento [231] resurfaces a user's interests when a similar activity recurs at a comparable place and time. Other systems poll on a schedule: Long-term TOA [315] passes its trigger conditions to a periodic background monitor, and OpenClaw's [332] heartbeat runs periodic check turns that stay silent when nothing needs attention.[1] A condition can also be checked again just before delivery. Orchid [313] reminds only if the watched condition still holds, and the MoveCare [158] home robot checks each week whether the user has already measured their weight before reminding. Feedback can narrow the watch, as when a user tells Caddy [351] that an alert was not useful. Benchmarks for these checks measure this trade-off. In PM-Bench [324], a to-do ledger of pending intentions gives the highest precision, the share of actions taken that are correct. Fixed 30-minute heartbeats, in contrast, catch the most updates but add many false alarms. TriggerBench [325] finds that stronger reasoning improves recall but can drift toward reminding every time.

4.3.4 How Records Change: Consolidation and Correction

Consolidation. Consolidation turns repeated episodes into summaries, preferences, or procedures [289, 293]. Mobile-Agent-E [136], for example, derives Tips and executable Shortcuts from completed tasks, which can speed up later execution. Because Shortcuts are executable code, writing them also revises the mechanism. Some designs link each consolidated record to the episodes it came from, or hold new episodes in a buffer before merging them [352, 336, 342, 353, 354], so the supporting evidence stays available as records change. Methods also differ in when they consolidate. LightMem [286] groups topics and defers maintenance, and sleep-time processing [287] maintains records between requests, which keeps this work out of response time. Consolidation also has a cost. Controlled update studies [355] and LifeDialBench [356] find settings where rewritten summaries are less useful than preserved episodes. A consolidated reminder preference, for example, saves repeated interpretation, but the individual dismissals are still needed to tell a recurring objection from one busy afternoon. The saving is worth it only when the summary keeps the distinctions that later decisions

[1] OpenClaw heartbeat: https://docs.openclaw.ai/gateway/heartbeat

depend on. During consolidation, an always-on assistant still needs a usable record and a way to carry later corrections into the summaries it produces.

Correction. Correction depends on why a stored value needs to change. If the user's situation changed, as when they move, the new address replaces the old one, but the old one was true when it was recorded, so versioning keeps it as history. If the record was wrong from the start, as when a name was misheard, it can be amended in place. MemConflict [321] draws similar distinctions among temporal, factual, and contextual conflicts. When it is unclear which value holds, overwriting the older record can discard the correct one, and asking the user can settle the question before a pending action uses either. A commitment can also change while the assistant waits to carry it out, and its trigger must then change accordingly. In PM-Bench [324], pending tasks are rescheduled, cancelled, or tied to a different event, and acting on the old version counts as an error. Even the best of the tested setups carries out fewer than half of the updated tasks at the right time. Updating one record, however, does not repair the records derived from it. StateAuditor [357] shows that summaries, later responses, and cached reads, which reuse earlier lookups, can still depend on the old value. An always-on assistant can therefore remember that the user corrected a preference yet still fail to follow it in the next interaction.

4.3.5 What Stops Influencing Later Decisions: Retirement, Forgetting, and Deletion

Ending validity and ending access. We distinguish the ways in which records end. Some end a record's validity: a meeting time expires once the meeting has passed, and a new time supersedes the old one. Others end access: redaction removes only the private details of a record, and deletion removes the whole record. When a record is removed, the store can keep a small marker in its place, called a tombstone, so that the revoked record is not imported again. The choice follows the user's need: completing a task can retire its pending commitment while preserving its outcome, whereas a deletion request can require removing the content. Repeated failures can also make a reusable procedure obsolete. After each task in a stream of browser tasks, PANDO [288] distills reusable rules and parameterized routines, procedures that take new inputs at each use. It demotes repeatedly failing skills and adds them to a persistent blacklist that blocks matching candidates from re-entering the skill library. Retirement thus controls both continued use and later readmission.

Closing fulfilled commitments. A fulfilled commitment stops producing reminders when the assistant closes it. The evidence for closing can come from several places: the user's report, a confirmation the assistant requests, or a verified payment confirmation. CareCall's [311] long-term memory removes a pending health item once the user reports that the treatment is complete, and Folk [358] asks for a screenshot or a one-line confirmation before closing a task. Pally [312] closes a bill reminder when a verified payment confirmation arrives, which needs no report from the user. The hardest case is a completion that leaves no trace. MoveCare [158] notices a completion the user never reports only when a measurement reaches it, and GapChat [317] instead estimates how a user's ongoing events progress from the time elapsed between sessions. Retiring a commitment when completion evidence is incomplete or ambiguous remains largely unevaluated. Cancellation and replacement are the other routes to retirement. The Prospective Intention Store [316] records done and cancelled states, and StateMemBench [359] tests whether memory systems follow commitments that are later retracted or superseded. IPIBench [129] asks a streaming assistant to cancel a monitoring request and stop responding to its target.

Removing residual influence. Information from a deleted source can remain in a summary, index, cache, or derived procedure, and source links help locate derived records that need removal or recomputation. Memora [337] evaluates whether invalidated memories still affect later answers and recommendations. A survey of long-term memory security likewise ties such leftover information to provenance and to forgetting at each stage of a record's life [37]. If the evidence also supported a mechanism revision, deleting the retained content and rolling back its change to θ_t are separate operations. Deletion is therefore complete only when later decisions no longer depend on the removed information, not when one store no longer holds it.

4.3.6 Learning the Memory Policy

Beyond any single decision, we ask how the Memory policy itself can be learned and revised. MemSkill [360] separates the records of individual interactions from skills that describe how memory operations are carried out. A trained controller selects skills and a fixed executor applies them. A second language model acts as a designer: it revises the skill bank from difficult training cases and can roll back to stronger earlier versions. Changing an operation skill alters how future records are handled, so it revises the mechanism, and the same change made from the assistant's own use would count as self-evolution. AdaMem [299] likewise revises its natural-language write policies from periodic feedback in simulated interactions. PANDO's blacklist differs: it retires failing browser-task skills by fixed criteria, which changes what the agent can do but leaves the Memory policy unchanged. Personal assistants rarely report revising a memory rule from the outcomes of their own deployment.

These Memory decisions leave two gaps. In the reviewed designs, history search starts from a query or a predicted need, and none gathers the context of a pending commitment before it falls due. For example, no design gathers the boarding pass and the hotel booking before the assistant reminds the user to leave for the airport. Corrections stop short in a different way: rules written as code from earlier records, like summaries and cached reads, can keep the old value. Scheduling retrieval by the commitments Memory already holds, and linking each derived record or rule to its source, would narrow these gaps.

4.4 Memory Coupling Across the Loop

The Memory policy decides what to do with new content and with the records already kept. This section turns to where new content comes from. Memory receives two kinds of input: what Sensing observes, including later feedback, through $S \to M$, and what Action did, with any immediate result, through $A \to M$. Both serve the two loops. The continuing loop keeps events and commitments from $S \to M$ and action records from $A \to M$ so that work can resume, and a self-evolving loop traces later feedback back to the action records behind it. Table 12 lists reported examples for both channels and what each study evaluated.

4.4.1 Local Decisions and Coupling Boundaries

Memory's local policy decides what to retain, update, or retire, but these decisions depend on what crosses its coupling boundaries from Sensing ($S \to M$) and from Action ($A \to M$). An update can therefore fail because the right input never reached Memory, even when the decision itself is sound. Table 11 separates Memory's own decisions from the information it receives through these two channels. The two channels carry different content, and an effective update often needs both. In the running example, the user's remark that the report has been sent arrives through $S \to M$ but does not say which reminder it concerns. The record of the sent reminder cannot confirm that the user acted on it either. Memory retires the commitment only by matching the remark to that record, so for feedback, finding the earlier decision it concerns is a prerequisite for the update.

Table 11 Local Memory decisions and incoming information.

Relation	Description	Example
Local policy	Selects and maintains stored records	Merge duplicate notes, preserving sources
$S \to M$	Sensed content is proposed for retention	A spoken commitment becomes a prospective record
$A \to M$	Action context and immediate results are retained	A reminder keeps its context and delivery status

4.4.2 Sensing to Memory: the $S \to M$ Channel

Selecting observations for later use. We first ask which details of an observation Memory keeps for later decisions. For a spoken commitment, the details that matter are the speaker, the intended action, and any timing or completion conditions. Commitments often first appear in what Sensing observes. Omi [138] turns action items from recorded conversations into tasks, and StreamMemBench [361] lists commitments and

Table 12 Information supplied to Memory through its two incoming channels. Rows connect the supplied information to the update it supports and to the reported evaluation.

Work / protocol	Supplied information	Update supported	Reported evaluation	URL
S→M: observations selected for retention				
AdaMem [299]	Personal relevance	Write selection	Personalized writing and use	/GH
PerMem-Bench [338]	User-specific context	Storage admission	Personalized storage decisions	/GH
Imprint [193]	Personal event records	Record construction	Compression and later recall	/–
Infini Memory [342]	Sources and topic state	Topic documents	Maintenance and subsequent use	/GH
ProEvent [335]	Commitments in chats	Timetable	Response timing, cancellations	/–
A→M: action context and outcomes retained for later use				
AssistantX [2]	Expected, observed state	Memory and plans	Reflection ablation, office test	/GH
KnowAct-GUIClaw [239]	Subtask results, feedback	Subtask records	Combined additions, skill test	/GH
MemQ [353]	Delayed utility	Linked records	Provenance-based propagation	/GH

plans among the evidence a streaming assistant should retain. Users can also hand such tasks to the assistant directly, as when they tell the ElliQ [362] companion robot about upcoming appointments.[1] Whatever the source, a suggestion, an accepted commitment, and a report of completion can mention the same task but call for different updates. Keeping the three apart lets Memory decide whether to create, refresh, or retire a prospective record, even when the original conversation is no longer available. The details kept when an observation enters Memory therefore limit which updates are possible later.

Preserving source and permission. We then ask whether an observation's source and permission survive the transfer. Without its source, a record cannot be rechecked or corrected later. Permission controls act at two points: MemPrivacy [334] masks sensitive values before content leaves the device, and Windows Recall [139] encrypts its stored snapshots and gates access to them. Neither handles sensitive details that the masking misses.

Why observation-to-memory transfer is difficult. First, **Memory must decide what to keep before later needs are known**. Second, **the right repair depends on where the information was lost**. New evidence can complete a partial record, but stricter admission cannot repair a summary that has already lost its source. A note that the user prefers mornings, for example, cannot be rechecked when the preference changes if it has no link to the conversation behind it. To locate such losses, AMemGym [363] separates writing, reading, and utilization, and WorldMemArena [364] combines multimodal items to remember, later updates to them, and distractors, irrelevant items mixed in with useful ones. Both distinguish failures to keep useful evidence from failures to use it, a distinction needed when evaluating S→M.

4.4.3 Action to Memory: the A→M Channel

Retaining action context for later use. Action passes useful information to Memory, so that the assistant can later resume a task after an interruption, avoid repeating the same reminder, and connect an outcome to the decision it concerns. This information is its action context: what the assistant intended, what it decided and why, what it did and promised, and any outcome already known. Suppose the assistant has just emailed a colleague to ask for a meeting time. Action then passes on why it chose that moment and that the email went out. The sending is now done, while the user's task stays open until the colleague replies. The assistant's own promises are part of this context too. In the PITCH [248] field study, the agent promised a reminder that it had no long-term memory to deliver, whereas Pally [312] sets a real reminder whenever it promises to follow up.

Matching later feedback to the record it concerns. Feedback often arrives after the action has ended. It then reaches Memory through the S→M channel, and Memory must find the record it concerns. This is easiest when the feedback names that record. An explicit correction does so directly, since it states which commitment changed. Some outcomes also name their target, as when a payment confirmation identifies the

[1] ElliQ FAQ: https://elliq.com/pages/faqs

bill that a reminder concerned [312]. A dismissal is less clear: it identifies the suggestion but not the reason for dismissing it. A silent completion names nothing at all, as when the user finishes a task without telling the assistant. To link such feedback, other work [365, 366, 367, 368] connects outcomes to stored experience or memory operations. When evidence of a record's usefulness arrives late, MemQ [353] passes it along the links between records and their sources.

Why action-to-memory transfer is difficult. First, **feedback often arrives late or reveals only part of what happened**. In an always-on assistant it often follows several interventions. Matching is hardest when decisions overlap, as when two reminders and a calendar change precede the same outcome. Learned memory managers face a simpler form of this problem. Memory-R1 [366] rewards its memory operations only by whether a later answer is correct, so it has to learn which of many earlier operations mattered. Second, **revising a rule is harder than correcting a record**. Memento [231] and ProAgent [130] correct records within their current rules. ProPerSim [369] and Time2Stop [370] instead retrain their model or policy, so an outcome linked to the wrong decision would carry into later ones. When a faulty record also shapes how later feedback is read, or stops the assistant from looking for evidence against it, each revision can preserve or amplify the original mistake, a risk we call a *memory trap*. In RoMeRL [371], rewards for successful tasks can reinforce memories that did not contribute to success, and its method mitigates this without fully resolving the attribution. No reviewed always-on system yet shows lasting protection against the trap.

4.5 Evaluation

4.5.1 Evaluation Metrics

A memory error can surface long after the record was written, so evaluation has to follow records beyond the session that wrote them. Table 13 lists the benchmarks and evaluation settings for memory, grouped by the operation they test. We arrange their metrics by following a record through its life. The first question is whether it enters the store: extraction recall measures whether relevant evidence is written at all. The next is whether it stays valid, which update accuracy and omission rates check as later changes arrive. Then comes use: retrieval precision measures what is returned, and downstream task results measure whether it helped. The last is whether it stops influencing behavior when it should, which invalidation tests check. Commitments add their own measures: timely recall, false alarms, and whether fulfilled or cancelled commitments stop producing reminders. Write and read latency, storage growth, and correction effort capture the costs of keeping records. Reading more evidence also takes time. In LongMemEval-V2 [372], removing one agent's pool of raw records lowers its answer accuracy on the benchmark's Small subset from 58.6% to 42.3% and its query latency from 26.9 to 16.7 seconds, and agent-native assessment [373] likewise compares answer quality with operation latency across memory systems. Results also depend on the order of events, since a query asked before a correction is answered from the old value and one asked after it from the new one.

4.5.2 Benchmarks and Evaluation Settings

Recall baselines. Recall baselines are the first group in Table 13: they supply a history and then ask about it. They test recall, updates, and forgetting, but not the handling of due commitments. LoCoMo [374] and LongMemEval [375] make conversational recall, temporal reasoning, and updates comparable, and MemoryAgentBench [376] extends recall to incremental agent memory. To show what each representation loses, MemLens [377] and MemEye [378] compare visual input with text, captions, or oracle evidence, which hands the relevant evidence to the model directly. Over a week of egocentric video, EgoMemReason [379] finds that the best model answers 39.6% of its questions but 67.2% when given only the relevant clips, so finding the evidence is a large part of the failure. Embodied settings ask the same of a robot: H-EMV [291] answers questions over a robot's history of up to two months, and FindingDory [380] scores whether a robot can navigate back to objects that were moved.

Updating records as input arrives. Some benchmarks test how well a memory system updates its own records as new input arrives. HaluMem [381] scores extraction and then updates, since a memory can update only what it has extracted. It separates two kinds of error: hallucinated updates, which the input does not support, and omitted updates, which the input calls for but the memory never makes. Its Long

setting adds irrelevant dialogue to Medium, increasing both length and interference. For the tested Mem0 configuration, the Long setting lowers extraction recall sharply and raises update omissions to almost all real updates, while update hallucinations fall almost to zero [381]. Low hallucination therefore does not indicate reliable maintenance. MemOps [382] scores responses against labeled memory transitions, annotations of how memory should change. LifeDialBench [356] follows lifelog memory, built online from a running record of daily life, through successive updates. MeMoREval [383] asks a robot to decide what to store before it knows the later tasks and counts the tokens spent writing and reading memory. MemPrivacy [334] shows another cost: masking sensitive information can also remove evidence that a later answer needs.

Keeping what holds valid. Another group of benchmarks tests whether stored state stays valid [233, 321, 337]. For longer sequences of changes, MINTEval [384] tests whether agents can still recall a fact after many updates, when old and new versions interfere, and whether they can combine several such facts. StateMemBench [359] tracks a user's state across sessions as the user declares rules, makes temporary exceptions, states commitments, and retracts earlier statements. WorldLines [385] brings such changes to embodied settings, asking about overwritten object and device states, preferences, and routines across several days of simulated household activity. Longitudinal studies add sources of error that one-shot tests miss: summaries can hide contamination [386], and a committed claim can sustain sycophantic behavior across interactions [387].

Holding what is pending. PM-Bench [324] spreads event-based and time-based commitments over a simulated week. On it, a small model with the typed Prospective Intention Store [316] scores about ten times higher than with the best of seven retrospective memory methods, and a larger model gains further with the same store. The gap supports treating pending commitments as a class of their own, separate from retrieved history. ProEvent [335] scores timetable maintenance from ongoing chats, and the tested agents often act when they should not and mishandle cancellations expressed only implicitly.

Testing later use after change. Benchmarks of later use ask two questions: **whether retained state helps later tasks**, and **whether a revised mechanism makes later tasks better**. Most address the first and show that earlier experience can hurt later decisions as well as help them. MemoryArena [23] and MEMENTO [388] make later success depend on memory built earlier, the latter for household robots. PraxisMemBench [389] extends this question to user-conditioned procedures: personal context, reusable procedures, and their associations change across sessions, and later executable tasks require their joint use. Its histories and users are synthetic, and each session begins with a request, so it tests how memory is maintained, not assistance that starts between requests. PATH-Bench [390] and CL-Bench [391] vary the history before a task, personal-context suites widen histories and tasks [392, 236, 393, 394, 345, 395, 396, 397], and LifelongAgentBench [398], VitaBench 2.0 [399], and HealthClaw's synthetic tests [303] let state change over time. Fewer come close to the second question, the last step in verifying self-evolution after record, outcome, attribution, and revision. EvoMemBench [400], BenchTrace [401], and ContinualSkillBench [402] test whether reused experience and skills improve later tasks, and PAST-Bench [24] switches retained records and saved skills on or off in fresh sessions to check that the gains run through them. A direct test would run the same situations before and after a revision that comes from the assistant's own use, not from a designer's change between runs.

> **Faulty Memory Can Trap the System in Repeated Mistakes**
>
> **A wrong belief can spread before anyone notices.** Methods that mark outdated records need new evidence, so a belief that turns wrong silently keeps being reused in summaries and commitments.
>
> **Keeping records valid is harder than retrieving them.** An update can be missed, and even a successful one may not reach the summaries and cached reads built on the old value.
>
> **Commitments need a store of their own.** Whether they come from a request, a conversation, or the assistant's own promise, pending commitments must come back on their own when their condition holds and must record whether they are still pending, done, or cancelled. Retrieved history does neither.

Table 13 **Memory benchmarks and evaluation settings.**

Benchmark / protocol	Evaluation setting	Evaluated content	Framework	URL
Recall baselines				
EgoMemReason [379]	Week-long egocentric video	Entity, event, and behavior memory	S → M, $\mathcal{B}$	
LoCoMo [374]	Multi-session conversations	Long-term conversational recall	S → M, $\mathcal{B}$	
LongMemEval [375]	Multi-session histories	Temporal reasoning, updates, abstention	S → M, $\mathcal{B}$	
MemoryAgentBench [376]	Incremental agent memory	Retrieval, learning, forgetting	S → M	
MemLens [377]	Multimodal memory tasks	Image-removal and oracle controls	S → M, $\mathcal{B}$	
MemEye [378]	Visual evidence across sessions	Text, caption, and oracle-image controls	S → M, $\mathcal{B}$	
RNG-Bench [403]	Remembering-to-act tasks	External-state and history controls	S → M, M → A	
LongMemEval-V2 [372]	Long agent histories	Environment knowledge and query latency	A → M, c_t: res., $\mathcal{B}$	
HippoCamp [393]	Personal file systems	File-grounded facts and user profiling	$\mathcal{B}$	
H-EMV [291]	Robot life histories	QA over months of experience	S → M, $\mathcal{B}$	
FindingDory [380]	Long household video	Recall of moved objects	S → M, M → A	–
ReMEmbR [143]	Campus robot videos	Spatial, temporal, descriptive questions	S → M	
Admission: what enters the store				
LifeDialBench [356]	Online lifelog memory	Event and temporal recall	S → M, $\mathcal{B}$	
HaluMem [381]	Extraction, updates, and QA	Hallucinations and omissions	S → M, $\mathcal{B}$	
AMemGym [363]	Interactive memory operations	Writing, reading, utilization	θ_t; S → M, M → A	–
PerMem-Bench [338]	Personalized storage	Future-relevant content selection	S → M, c_t: res., $\mathcal{B}$	
MemOps [382]	Conversations with embedded memory operations	Trigger, target, and state-transition accuracy	S → M, M → A, c_t: priv.	
MemPrivacy [334]	Sensitive content in memory	Masking and retained memory utility	S → M, c_t: priv.	
WorldMemArena [364]	Multimodal memory updates	Writing, updates, retrieval, and use	S → M, A → M, $\mathcal{B}$	
MeMoREval [383]	Events before unknown tasks	What to store, and its cost	S → M, M → A, c_t: res., $\mathcal{B}$	
Profile: keeping what holds valid				
STALE [233]	Outdated personal records	Stale-memory recognition	S → M, M → A, $\mathcal{B}$	
MemConflict [321]	Conflicting memory records	Conflict handling	S → M, $\mathcal{B}$	
Memora [337]	Invalidated memory	Invalidation-aware forgetting	S → M, M → A, $\mathcal{B}$	
MINTEval [384]	Repeated multi-target updates	Recall and reasoning under interference	S → M	
StateMemBench [359]	Evolving state operations	Current versus stale state	S → M, $\mathcal{B}$	–
PASB [387]	Two-session agent workspaces	Saved user claims in later answers	S → M, M → A, $\mathcal{B}$	
WorldLines [385]	Multi-day household traces	Overwritten states and routines	S → M, M → A, $\mathcal{B}$	–
Prospective: holding what is pending				
TriggerBench [325]	Latent constraints	Timely recall without false alarms	S → M, M → A, $\notin \mathcal{R}$	–
PM-Bench [324]	Simulated weekly activities	Due, updated, and cross-day commitments	M → A, A → S, $\mathcal{B}$	
ProEvent [335]	Ongoing chat threads	Timetable upkeep and cancellations	S → A, S → M, M → A, $\notin \mathcal{R}$	–
Coupling and later use: state influences subsequent decisions				
StreamMemBench [361]	Streaming input and feedback	Evidence use and follow-up reuse	S → M, M → A, A → M, $\mathcal{B}$	
MemoryArena [23]	Interdependent subtasks	Memory construction and action use	M → A, A → M, $\mathcal{B}$	
PAST-Bench [24]	Cross-session persistence pairs	Later task gain or decline	θ_t; M → A, $\mathcal{B}$	
PATH-Bench [390]	Helpful/interfering histories	Transfer and forgetting	M → A, A → M, $\mathcal{B}$	–
CL-Bench [391]	Stateful task sequences	Learning gain over prior capability	A → M, c_t: res., $\mathcal{B}$	–
EvoMemBench [400]	Within- and cross-episode tasks	Knowledge versus execution memory	S → M, A → M, $\mathcal{B}$	
BenchTrace [401]	Annotated failure episodes	Reflection quality and failure avoidance	θ_t; A → M, $\mathcal{B}$	
ContinualSkillBench [402]	Ordered, interdependent tasks	Skill reuse versus context-only learning	θ_t; A → M, $\mathcal{B}$	
VitaBench 2.0 [399]	Life-service task sequences	Preference extraction, drift, and use	S → M, M → A, $\mathcal{B}$	
DynamicMem [345]	15-month multi-app logs	Evolving profile recovery and use	S → M, M → A, $\mathcal{B}$	
LifeVal [404]	Writers' chats and memories	Personal context used in requested tasks	M → A, $\mathcal{B}$	–
OP-Bench [349]	Dialogue memory probes	Irrelevance, repetition, sycophancy	S → M, M → A, $\mathcal{B}$	
LifelongAgentBench [398]	Sequential DB/OS/KG tasks	Later success from replayed trajectories	M → A, A → M, $\mathcal{B}$	
MEMENTO [388]	Two-stage household tasks	Memory use under vague instructions	M → A, $\mathcal{B}$	

Framework: $X \to Y$ denotes a coupling channel; c_t gives the shared conditions that are scored; $\notin \mathcal{R}$ marks a scored decision at a step outside $\mathcal{R}$, where no explicit request is being carried out; $\mathcal{B}$ marks state that is retained when an interaction ends and is used later; θ_t marks a protocol that scores a mechanism revision. A channel is marked only when the score of the protocol depends on that flow between modules. Condition abbreviations: **priv.** privacy, **res.** resources.

5 Action

Action decides whether, when, and how the assistant acts on the external world, for example by sending a message, editing a file, or speaking aloud. In real use, opportunities to act arise continuously without a request, and staying silent is often the right choice. Many actions cannot be undone once taken, and an untimely intervention spends attention the user cannot get back. Because the assistant cannot ask for permission at every step, the user steers the loop by setting its direction and limits in advance. When an action would exceed those limits, such as an irreversible step the user has not approved, the assistant pauses and asks. Action's main role is to provide the *operational continuity* of §2.1, keeping the assistant helpful between requests. At a step that no request starts, Sensing and Memory supply context, but Action decides whether to act, so we classify actions by what starts them. Reactive actions answer a current request, deferred actions carry out an earlier commitment, and proactive actions respond to a newly detected need. An action can also leave a commitment behind, and its success or failure can surface only days later. A self-evolving loop uses the record of these delayed outcomes to revise when to intervene, when to wait, and when to ask. Figure 9 shows how this section is organized.



Figure 9 Action section structure and taxonomy.

5.1 Action Space

The action space covers the choices available to the assistant at each step and the format each action takes. Notifying the user, drafting an email, and sending it require different evidence and permissions and create different commitments. Choosing among them weighs the expected benefit of each action against its cost to the user, such as an interruption. Between requests, the choice also accounts for the value of waiting and for the attention, access, and recovery costs that accumulate across interactions.

Following the two parts of a module's space, §5.1.1 classifies actions by what starts them, into **reactive**, **deferred**, and **proactive** action, and §5.1.2 describes the format an action takes: the interfaces through which it takes effect, its consequences, and the feedback it returns. Waiting is available in each class, and one episode can lead to another: a proactive suggestion that the user accepts becomes a commitment that later starts a deferred action.

5.1.1 What Starts an Action

Reactive action. Table 14 lists the different classes of action. Reactive action responds to the user's request and is the most common class. Every step of the episode lies inside $\mathcal{R}$ of §2.1, including the request's continuing execution, however long it runs. Task-execution benchmarks measure reactive action, the baseline that the other two classes extend. In an always-on assistant, a request can draw on what the assistant observed before it was made. LightMem-Ego [137] captures egocentric video and audio in the background and answers later questions, such as where an object was left or what was said in a conversation. M3-Agent [95] builds long-term memory from continuous video and audio and retrieves it when given a task. VisionClaw [10] acts only when asked: it keeps smart-glasses perception available and turns what the user sees into digital operations, such as creating calendar events from posters. Requested work can also outlast the user's attention: Muse, Dot, and Grok Bot keep working on their own cloud computers after the user closes the app [15, 16, 242]. When such work pauses for clarification or permission, as when VisionClaw confirms the recipient and content before it sends a message, it remains reactive action for the same request when it resumes. A request can still leave work for later, and that work becomes a commitment.

Deferred action. Deferred action carries out a task the assistant undertook earlier, at a later moment and without a new request. The task is held as a commitment, and the action starts when the commitment is reactivated, as with a reminder after a meeting, a weekly summary, or a follow-up when a reply arrives. The commitment can come from three sources. First, **explicit rules** can be set up in conversation: RuleBot++ [413] creates and modifies smart-home trigger-action automations, which run an action when a specified condition holds. Products make such rules routine. Manus [408], Claude Cowork [410], and ChatGPT Tasks [298][1] save a task's instructions and run them on a schedule, and ChatGPT Tasks can also start one when a supported Gmail, Slack, or GitHub event arrives. Brea [409] runs automations the same way but messages the user only when a run finds something new. Second, **vague goals** must first be turned into plans: Sasha [415] turns under-specified goals into device plans or automation routines, and SAGE [133] keeps monitoring the state of home devices and acts once a set condition holds. Third, for **commitments the assistant infers**, it writes the trigger itself: Long-term TOA [315] formulates trigger conditions from the dialogue and follows up when the environment changes, and Pally [312] can offer a standing rule after the user approves the same action several times. IPIBench [129] tests how agents handle standing requests that users add, modify, or cancel while a video stream continues. A deferred action still has to survive interruptions and meet the current conditions, including authority, when it runs.

Proactive action. Proactive action is help that nobody asked for. The assistant notices a need on its own, such as a calendar conflict or a document the user will need in the next meeting, and decides whether to offer help. This takes two decisions. First, **detecting the need** comes from learning when unsolicited help is welcome: Proactive Agent [12] and Alexa Hunches [426] learn this from event streams. ProAct [152] uses idle time between requests to prepare for likely upcoming needs. Second, **deciding when to speak** separates the timing of help from its content: LlamaPIE [416] separates the decision to respond from generating the response. In streaming video, MMDuet [430], MMDuet2 [417], and StreamBridge [21] decide when to respond. Speak or Stay Silent [431] and When2Speak [432] test when an assistant should join an ongoing conversation. Field studies such as ProAIDE's [13] five-day deployment and the six-week HeartSteps [433] trial show how these decisions play out over days. Deciding for itself whether to act on a newly detected need is *operational continuity*, and revising that decision rule from outcomes is where self-evolution enters.

[1] ChatGPT scheduled tasks: https://learn.chatgpt.com/docs/automations

Table 14 **Representative actions by what starts them.**

Work	Group	Key feature	Interface	URL
Reactive action				
VisionClaw [10]	User request	Glasses view to digital tasks	Smart glasses + apps	↗/
LightMem-Ego [137]	User request	Answers from background capture	Phones and glasses	↗/
M3-Agent [95]	User request	Uses long-term memory for tasks	Video + audio stream	↗/
Egocentric Co-Pilot [151]	User request	Web tasks from glasses requests	Smart glasses	↗/
Dot tasks [16]	User request	Requested task runs after app closes	Cloud computer	↗/–
Muse tasks [15]	User request	Sentinel checks risky steps	Cloud computer	↗/–
Grok Bot tasks [242]	User request	Confirms requested visual changes	Cloud computer	↗/–
UI-TARS-2 [405]	User request	Multi-turn GUI agent	GUI + tools	↗/–
KnowAct-GUIClaw [239]	User request	Resumes checkpointed work	GUI execution	↗/
Hobbit [3]	User request	Verifies each grasp, retries once	Home robot	↗/–
ABot-Claw [6]	User request	Critic corrects and replans	Robots	↗/
KnowNo [235]	User request	Asks for help when unsure	Robot planner	↗/
Robix [406]	User request	Clarifies, takes mid-task corrections	Robot planner	↗/–
MICoBot [407]	User request	Negotiates who does each step	Mobile manipulator	↗/–
Deferred action				
CareCall [311]	Explicit rule	Weekly calls with retained context	Phone calls	↗/–
ChatGPT Tasks [298]	Explicit rule	Scheduled or event-started tasks	Connected apps	↗/–
Manus [408]	Explicit rule	Scheduled task runs	Cloud sandbox	↗/–
Brea [409]	Explicit rule	Reports only new findings	Connected apps	↗/–
PITCH [248]	Explicit rule	Morning plan, evening check-in	Slack	↗/–
Claude Cowork [410]	Explicit rule	Scheduled tasks on local files	Desktop app	↗/–
Hermes cron jobs [333]	Explicit rule	Missed slot runs once after restart	Local agent	↗/
EAIA [411]	Explicit rule	Drafts wait for approval	Email	↗/
Gemini schedules [412]	Explicit rule	Scheduled prompts over apps	Connected apps	↗/–
Inbox Zero rules [314]	Explicit rule	Rule revised after a wrong match	Email	↗/–
Long-term TOA [315]	Inferred commitment	Triggers written from dialogue	Dialogue	↗/–
Pally automations [312]	Inferred commitment	Rule after repeated approvals	Messages	↗/–
RuleBot++ [413]	Explicit rule	Automations set up by chat	Smart home	↗/–
MoveCare [158]	Explicit rule	Checks weight before reminding	Home robot	↗/–
Family reminders [318]	Explicit rule	Reminds when scene matches plan	Mobile social robot	↗/–
Hyodol [414]	Explicit rule	Twice-daily voice health check-ins	Social robot	↗/–
Sasha [415]	Vague goal	Goals turned into routines	Smart home	↗/–
SAGE [133]	Vague goal	Monitors device state	Smart home	↗/–
Proactive action				
AiGet [247]	Detecting the need	Surfaces knowledge in daily scenes	Smart glasses	↗/
SocialMind [246]	Detecting the need	Social suggestions on AR display	AR glasses	↗/–
LlamaPIE [416]	Deciding when to speak	Small model decides when to reply	In-ear audio	↗/
MMDuet2 [417]	Deciding when to speak	Decides when to respond	Streaming video	↗/
StreamBridge [21]	Deciding when to speak	Proactive streaming replies	Streaming video	↗/–
Sensible Agent [418]	Deciding when to speak	Picks content, modality, and moment	AR glasses	↗/–
ProMemAssist [9]	Deciding when to speak	Times help by working-memory load	Smart glasses	↗/–
EgoVoice [419]	Deciding when to speak	Listens or speaks each second	Egocentric stream	↗/–
PACE [420]	Deciding when to speak	Waits until the event ends	Streaming video	↗/–
Proactive Agent [12]	Detecting the need	Learns when help is welcome	Event streams	↗/
ProAct [152]	Detecting the need	Prepares during idle time	Text chat	↗/
ProAIDE [13]	Detecting the need	Help fitted to the workflow	Programming IDE	↗/–
Xiaoyi Suggestions [421]	Detecting the need	Fewer of what is dismissed	Phone	↗/–
ProActor [422]	Deciding when to speak	RL-trained intervention timing	Dialogue	↗/–
Time2Stop [370]	Deciding when to speak	Times overuse interventions	Smartphone	↗/–
DIAMANTE [423]	Deciding when to speak	Learns text type and timing	Text messages	↗/–
Oralytics [424]	Deciding when to speak	Prompt policy revised weekly	Toothbrush + app	↗/
MiWaves [425]	Deciding when to speak	Scheduled RL policy updates	Mobile app	↗/
Alexa Hunches [426]	Detecting the need	Flags unusual device states	Smart home	↗/–
ElliQ chat [319]	Detecting the need	Starts chats several times a day	Social robot	↗/–
RoboOmni [427]	Detecting the need	Intent from speech, sounds, video	Robot arm	↗/
GAP [241]	Detecting the need	Acts on anticipated goals	Simulated home robot	↗/–
PACT [428]	Detecting the need	Asks or acts from cross-day history	Simulated home robot	↗/–
Learning to Assist [429]	Deciding when to speak	Waits until the person is ready	Collaborative robot	↗/–

5.1.2 Interfaces, Consequences, and Feedback

Interfaces and observable effects. An action's format begins with the interface through which it takes effect. Interfaces fall into three groups by what they affect. First, outputs to the user, such as dialogue, notifications, visual overlays on the user's view, and touch signals, address the user's attention or knowledge. Second, digital operations through a GUI or API change digital resources [405]. Third, physical effectors are the parts of a device that act on the world, and they change location, viewpoint, or objects [434, 97]. The interface determines whether the assistant can offer information, prepare material, or commit a change to another service, and these differ in the permission they require, the feedback they return, and the cost of recovery when plans change or execution is interrupted.

Consequences of an intervention. An intervention can interrupt the user, disclose information, or change external state. A private draft is easy to undo, while a public question can expose sensitive context. Recovery depends on the effect: a sent message can be read before it is deleted, and deleting it can remove its record without reversing disclosure. Silence also has consequences: it can avoid interruption or leave an opportunity unmet. These effects can become visible after the interaction ends, and their cause and value can be unclear, since the user can complete a task independently while the assistant remains silent. Subsequent observations and directly delivered feedback inform the next decision.

Feedback from execution and interaction. Execution feedback reveals what happened: API receipts report operations, errors indicate failures, file differences show changes, and test results assess behavior. Return interfaces present execution feedback to the agent, and studies of them show that the supplied detail affects later performance [435]. User responses, in contrast, provide evidence about whether an intervention was welcome or useful. Alongside explicit corrections, behavioral signals can supply feedback: ImplicitRM [436] learns a reward model, which scores responses by inferred user preference, from clicks and copies. A correction can identify a problem, whereas a missing click or copy leaves more uncertainty because a satisfied user can also take no action. This feedback helps Action choose whether to continue, clarify, retry, or wait. Feedback that arrives after the interaction reaches Memory through $\mathrm{S} \rightarrow \mathrm{M}$, where the decision and execution context retained through $\mathrm{A} \rightarrow \mathrm{M}$ helps link it to the earlier decision.

5.2 Shared Conditions for Action

All six shared conditions bind Action, because each can make a possible action wrong to take now. We take them in turn: user authority, safety, privacy, resource budgets, and finally uncertainty and availability.

What is the assistant allowed to do right now? User authority limits which resources an action is allowed to affect, for which audience, at what time, and with what lasting effect. This authority comes from grants, the permissions the user has given, and Muse [15], for example, checks each operation that needs permission against the user's settings. A typical grant permits drafting but requires confirmation before a message is sent, and EAIA [411] leaves the emails it drafts in an inbox until the user approves them. Grants also change over time. A grant often widens only as the assistant acts as expected, since one unexpected action can cost the user's trust, and the user can let it expire or revoke it. Connected services, such as a mail provider, add limits of their own. Each change must reach queued and delegated actions, yet pausing Dot [16] stops only its current main task, not its delegated tasks or scheduled runs. What is permitted therefore depends on the current conditions c_t, not on a stored instruction or an earlier grant.

What must not happen, even with permission? Safety limits the harm an action can cause, and no grant can widen that limit. The people and objects around the assistant, the device's physical limits, and platform or legal rules set this limit, as the safety requirements for personal care robots do [437]. Claude Duet [17], for example, keeps some actions off-limits even with the user's approval, such as trading securities or moving money between accounts. A permitted action can therefore still be unsafe. A home robot asked to tidy up must not move its arm while someone is within reach, and HumanCLAW [438] scores collisions and disturbed objects alongside task success. On the web, ST-WebAgentBench [439] credits a task only when the agent also follows every applicable policy, and the tested agents reach less than two-thirds of their plain

completion rate. Because an injury cannot be corrected afterwards, the limit applies before the action takes effect, which matters most while the user is away and no one reviews the action.

What is an action allowed to reveal, and to whom? Privacy limits what an action is allowed to disclose and to whom. The same information can be fine on the user's phone but not read aloud in a shared room or delivered days later to a different audience, and a sent message can expose a third party. In a confidential meeting, the assistant is allowed to process speech but not to pass the transcript to a tool or another person. Current agents do not reliably keep these limits. PrivacyLens [440] finds that language-model agents put sensitive details into messages where the context does not call for them, even when prompted to protect privacy. In EIA [441], content injected into a web page leads an agent to type the user's personal data where an attacker can read it. Claude Duet [17] adds a dedicated classifier and safety prompt that check each message for leaks of sensitive information. They are designed to refuse the most sensitive tier, such as card numbers and government IDs, and to let health or financial details through only when the user has explicitly asked to send them. Because a disclosure cannot be undone, this limit, like safety, applies before the action takes effect.

What does an intervention cost the user? A permitted intervention still draws on the user's attention, a limited resource. Interruption research treats an interruption as a cost [442] and studies which moments disrupt less [443, 444, 63]. For an assistant that runs all day, these costs add up, and its own earlier reminders count among them. Users state part of this limit directly, for example as hours without notifications or, in Claude Duet [17], as what they want to hear about and how often, and the assistant has to infer the rest from what the user is doing.

How certain is the situation, and is the interface available? Weak evidence and an unavailable interface also limit what can be done now. The assistant does not always know the current state of the user or of a task. KnowNo [235] measures such uncertainty and lets a robot planner ask for help when several actions remain plausible, and π-Bench [445] scores whether agents meet requirements before the user has to state them. An interface can be unavailable because a calendar service is offline or the user is away from the device that would show the reminder. HeartSteps [433] sends a suggestion only when the participant is available, for example not driving, and Gaia2 [446] runs asynchronously, so its apps keep changing while the agent works. Because both limits can change within minutes, the choice between acting now, waiting, and asking has to be made again.

5.3 Action Policy

The local Action policy π_A governs how the assistant turns a user request, a due commitment, or a detected need into a concrete action. Deciding whether to act is only the first of its choices. As Table 15 summarizes, π_A makes four decisions that constrain one another: whether and when to intervene, how much control to keep, which interface to use, and which effect is authorized.

Table 15 Local Action decisions and representative works.

Policy	Main choices	Cost / trade-off	Representative works
Gating/timing	Wait/clarify/prepare/proceed	Misses / interruption	ProActor [422], ProAct [152], HeartSteps [433]
Control depth	Direct/plan/delegate/resume	Recovery / duplicates	Enoch [447], Hermes [333], EAIA [411]
Effector	Dialogue/GUI/API/physical	Cost / receipt quality	VisionClaw [10], SAGE [133]
Authorization	Proceed/confirm/withhold	Confirmation / overreach	Muse [15], AGrail [448], EAL-Bench [449]

5.3.1 Whether and When to Intervene: Gating, Silence, and Asking

Using an expressed or inferred goal. Before intervening, the assistant has to decide whether an action is worth taking now, and this decision starts with where the goal comes from. A request or a retained commitment supplies the goal. When the assistant detects an opportunity on its own, it also has to judge whether the inferred need is relevant. Research on dialogue systems that offer help or steer the conversation

without being asked [38, 40] studies this judgment. Even a given goal leaves the decision open, because the deadline and current task status still affect whether to notify, clarify, or close the commitment. An inferred need can instead call for checking whether the user is ready to receive help now, or for offering help before carrying it out. In both cases, the assistant weighs the expected value of intervening now against that of waiting. Estimating this value means predicting what follows each choice, which the same language model can do or a separate world model can support [32].

Waiting for a useful moment. The policy can also postpone assistance to a less disruptive moment. Speech and video studies evaluate when to stay silent, when to begin responding, and how to alternate speaking with the user [431, 432, 272, 450, 451, 420]. In software development, ProAIDE's [13] five-day study in a programming environment relates receptivity to the current workflow. Learning to Assist [429] makes the same choice for a robot, deciding whether to help or wait during collaborative assembly. Waiting can preserve the opportunity if a better moment comes before the deadline, but repeated postponement can instead make the content irrelevant. Recent interruptions and unresolved commitments therefore matter alongside response quality.

Clarifying, preparing, or stopping. The obstacle determines how to avoid acting at once. When information is missing, a question can resolve an ambiguous goal. Preparation can make a future response cheaper when timing is poor. When the next step is unsafe, stopping can prevent an uncertain action from continuing. Robix [406], RoboOmni [427], and PACT [428] examine clarification before acting, while agentic abstention [452] studies stopping during execution. Asking can also prepare later sessions. ATRBench [453] scores whether a long-lived agent asks now for a reusable preference that the current task does not need, and KnowU-Bench [392] hides the user's profile and scores whether agents elicit missing preferences and hold back after a refusal. Each question carries an attention cost, but how that cost accumulates over weeks of questions is rarely measured. When the assistant asks, the answer returns as a new observation.

Choosing a separate or shared timing model. Deciding when to respond costs computation while the assistant waits: a small timing model saves it, but missed cues delay responses or lose the moment. The timing decision can use a separate model or shared representations, the same internal information used to generate a response. At runtime, several systems select opportunities before expensive generation: ProActor [422], LlamaPIE [416], PRPF [124], STEPS [454], and trigger-and-routing models [147], which decide when processing starts and where it goes. StreamBridge [21] uses a separate activation model, which decides whether to start a response. MMDuet [430] instead uses one model for both timing and speaking: as it reads each video frame, two small output layers score how informative and how relevant the moment is, and the model generates text only when these scores are high enough. A separate activation model can change without revising the response model, whereas shared representations make timing and generation use the same video-language state.

5.3.2 How Control Is Retained: Single-Step and Multi-Step

Retaining control across steps. We distinguish two depths of control. A single-step policy makes one bounded decision and finishes, whereas a multi-step policy retains control across a plan, observes intermediate results, and recovers from errors along the way. ReAct [455] interleaves reasoning and action in this way. Retaining more control offers more chances to recover but requires more inference. Bounded-task studies find diminishing success gains as the number of steps allowed per task grows [456], and an always-on assistant also needs budget for background work and other commitments. Control can also be distributed, as when the scaffolding keeps task state in a manager and gives executors fresh context [457]. Such handoffs need current evidence: stale or adversarial input can mislead decisions [167].

Resuming an unfinished commitment. Control across interactions includes resuming unfinished work. A reminder or delegated task can remain pending while the assistant sleeps, a tool is unavailable, or the user changes plans. Memory retains the commitment and the completed steps, and Action uses the current task state and permission to resume, amend, cancel, or seek confirmation. Correct resumption neither repeats nor skips work, yet retrying a write after a timeout can duplicate its effect. In LIMBO [243], leading models

rarely duplicate a write when they can read the resulting state, but they duplicate it in most episodes when the request is still in flight or was delivered twice. Among the reviewed works, only the Enoch [447] runtime tests resumption in a long-lived agent. In one supervised workflow, it recovers from a planned interruption, rejects a stale reply, and records one completion. Recovery uses a checkpoint, a saved execution state from which work can resume. No reviewed evaluation tests missed or repeated runs of personal commitments across restarts.

Product documentation describes how missed and repeated runs should be handled. Hermes [333] runs a slot missed during a restart exactly once and never reruns one that already ran,[1] and OpenClaw [332] does not replay attempts that already have a durable result.[2] Pally [312] does not resend delivered results and lets follow-ups that missed their moment expire instead of arriving weeks late. EAIA [411] keeps the agent's state between steps, so it can pause for the user's answer and resume. A completion confirmed elsewhere retires the commitment. These cases link execution control to observation through $A \rightarrow S$, and through $A \rightarrow M$ they give Memory the action context it needs to maintain the task durably.

5.3.3 Through Which Interface the State Changes: Effectors and Receipts

Choosing an execution interface. Choosing an interface determines which operation can produce the intended effect and what feedback it returns. General orchestrators, systems that coordinate tools and models, learn to select them by monetary cost, response delay, and preference [458]. For operating a selected device in the home, SAGE [133] reads the device's API documentation so it can operate without device-specific code. The available interfaces differ in execution cost and returned evidence, and their availability also changes when a device or service becomes inaccessible. When no software interface can finish a step, a person can take it over. Fo [459], for example, can hand a task to a trained human assistant who works for its user.

Selecting evidence for the next step. If repeating an operation risks a duplicate effect and the returned feedback does not confirm completion, the assistant checks again, by observation or by asking the user, before retrying. The cost of the initial operation thus understates the work needed to continue safely. For spoken responses, completion also depends on what has actually played. When the user interrupts ChipChat [460], its audio player reports the playback position to the language model. The model then removes generated but unplayed content from its cached context. For $A \rightarrow M$, this distinction determines which part of a response can be recorded as delivered.

5.3.4 Which Effect Is Authorized: Proceeding, Confirming, and Withholding

Checking permission before an action takes effect. Before an action takes effect, the Action policy checks whether it is authorized and chooses among three options: proceed, ask the user to confirm, or withhold the action. The same check covers safety and privacy, but confirmation clears only some limits: the user can grant missing access or agree to share their own information, whereas no confirmation lifts a safety limit, so there the policy can only change or withhold the action. Permission also sets only a ceiling, and below it, weak evidence can lead the policy to a lighter option, such as a draft, a clarifying question, or silence. Systems implement the check in different ways. Muse [15] runs it in a separate agent called Sentinel, which compares each outside action with the user's settings and which the agent proposing the action cannot override.[3] AGrail [448] generates safety checks that adapt to each task and refines them across tasks, and formal policy enforcement [461] checks actions against explicit written rules. Rules stated only in the prompt protect less: CRAFT and τ-break [462] try to persuade agents to break tool-use rules and find that repeating the rules in the prompt prevents some violations but not all. An assistant that acts between requests therefore needs a check that runs at the moment an action takes effect.

Rechecking after a pause or handoff. When a deferred action runs, or a task resumes after a pause or handoff, the policy checks again, because two things can change in the meantime. First, the situation: Pally [312] checks whether the user has already handled a task before sending a reminder, and ProEvent [335]

[1] Hermes Agent cron jobs: https://hermes-agent.nousresearch.com/docs/user-guide/features/cron
[2] OpenClaw automations: https://docs.openclaw.ai/releases/2026.8.1/automations-and-scheduling
[3] Muse safety: https://research.meta.ai/blog/security-and-safety-for-ai-agents-our-approach-with-muse

tests whether agents that keep a timetable notice cancellations that nobody states explicitly. By contrast, Gemini's [412] scheduled actions reuse the location recorded when the task was created and prepare content early, so fast-changing details can be out of date by delivery.[1] Second, authority can expire: when a sign-in times out, Pally pauses the automation that needs it and asks the user to reconnect. ChatGPT's [298] scheduled tasks, in contrast, can run without approval prompts when the organization's policy allows it. Authority kept in memory can also drift or be corrupted. In EAL-Bench [449], executors end up acting on approvals that no longer hold, and AuthMem-Bench [463] finds that consolidating memory can turn a claim into a standing instruction with more authority than its source. Background runs can also write unvetted content into memory that later shapes what the user sees [464].

Commit-time authorization [465] addresses both changes for general agents. It checks permissions at the moment an action takes effect instead of during planning or approval, so an approval counts only if the underlying context and evidence still hold when the action is executed. Permission to draft a message therefore does not cover sending it once send access has expired or been revoked, and the policy can then show a preview, ask again, or cancel.

5.3.5 Learning the Action Policy

The Action policy can be learned or revised, but most reviewed learning precedes deployment. It takes place in simulated settings or on annotated dialogues instead of during deployment. For learning before deployment, Proactive Agent [12] learns welcome offers from simulated event streams. ProActor [422] trains intervention timing on annotated dialogues with reinforcement learning, which adjusts a policy using rewards from actions. Communication Policy Evolution [466] instead evolves communication prompts through simulated interactions with user personas for tasks with vague specifications.

These rules stay fixed once the assistant runs, while some mobile-health interventions revise their rule during deployment. Oralytics [424] matches daily batches of brushing data from a sensor toothbrush to earlier prompt decisions and revises its policy weekly. MiWaves [425] revises its reinforcement-learning policy on a schedule during a clinical trial. Because these systems revise their own policy during deployment and keep deciding under the revised policy, they count as self-evolution.

Among products, Caddy [351] narrows an alert rule after the user says an alert was not useful, and Inbox Zero [314] revises a rule after the user fixes a wrong match. Revision during use thus appears in a few narrow forms: scheduled updates within clinical trials, retraining on the user's explicit agreement as in Time2Stop [370], and product rules changed after an explicit user correction. Learning when to intervene from implicit outcomes of everyday use, within the user's limits, remains open.

5.4 Action Coupling Across the Loop

Action receives two kinds of input: the current situation from Sensing, through $S \to A$, and retained context from Memory, through $M \to A$, including the due commitments that start deferred action. Its own outputs to Memory and Sensing, through $A \to M$ and $A \to S$, are treated with those modules. Because $S \to A$ already feeds every decision above, we focus on $M \to A$: which personal context reaches Action, how it is presented, and whether it still applies. Table 17 compares implementations and their evaluated Action decisions.

5.4.1 Local Decisions and Coupling Boundaries

Action's local policy decides whether, when, and how to act, but these decisions depend on what crosses its coupling boundaries from Sensing ($S \to A$) and from Memory ($M \to A$). An action can therefore go wrong because of its inputs, even when the decision itself is sound. Table 16 separates Action's own decisions from the information it receives through these two channels. The two channels carry different content, and a good decision often needs both. In the running example, a meeting-end cue arriving through $S \to A$ makes the reminder timely, while a completion record arriving through $M \to A$ removes the need to send it. The local policy weighs both against the current conditions to choose the intervention. Labeling each input by the

[1] Gemini scheduled actions: https://support.google.com/gemini/answer/16316416

channel that delivers it also shows where to repair a failure: an input that never arrives, one that arrives but does not fit, and one that fits but is misused each call for a different fix.

Table 16 Local Action decisions and incoming guidance.

Relation	Description	Example
Local policy	Chooses the intervention and its timing	Wait while the user is occupied
S→A	Current observations guide the action	A meeting-end cue makes a reminder timely
M→A	Retained personal state guides intervention	A completion record suppresses a reminder

5.4.2 Memory to Action: the M→A Channel

Using personal and task context. Retained context affects whether to intervene and how to perform a task: a recent-reminder record can suppress redundant advice, and task history can guide resumed work. In both cases, π_A combines the delivered guidance with current observations and permission. General GUI agents supply selected, compressed task history to each decision [467]. MEMENTO [388] gives agents correct perception and motor skills in simulated household tasks, so its results isolate how personal memory guides decisions. For intervention decisions, User as Code [302] selects alerts from typed personal state, and PASK [468] and PersonalAlign [469] combine long-term personal context with detected intent to offer proactive help. Both the record and the moment matter: in MemoProAgent [470], a language model gives one correct reminder at the right time in 96% of clips when it receives the relevant record only while the reminder is due, in 38% when it must choose the moment itself, and in 15% when the record is retrieved instead of given.

Delivering due commitments and procedures. A pending commitment reaches Action when its stored trigger holds, as in Long-term TOA [315] and the Prospective Intention Store [316]. Memory decides when a commitment is due, but Action still decides whether, when, and how to act on it: remind now, wait for a better moment, or close it if the task is already done. Memory can also deliver reusable procedures as context, such as the procedures and state cards that MMSkills [471] uses alongside a screenshot. A procedure's conditions, however, can stop holding when a paused task resumes or the application state changes.

Separating retrieval, delivery, and use. A failure of M→A can occur at three steps, each with its own repair: Memory never retrieves the right record, it delivers an irrelevant one, or Action applies the right record wrongly. *Demystifying Agent Skills* [472] measures these steps separately, as ranking, selection, and execution, when it studies how agents reuse skills. To test only Action's use, an evaluation can change the delivered context while keeping the store fixed, since replacing the store also changes retrieval. An unchanged response then means that current evidence already suffices or that the records add nothing new. The split also decides what a revision should target: the Memory policy for which guidance arrives, or π_A for how Action uses it.

Why memory-guided action is difficult. A record that Memory delivers does not always fit the decision at hand, for two reasons. First, **a record can conflict with the current situation**. An old quiet-morning preference, for example, can conflict with a nearing deadline. The PACE [473] benchmark evaluates whether agents retrieve the personal facts and events that reveal such a conflict with the request. Second, **a correct record can still be the wrong one to use**. A valid fact can be irrelevant to the present task. Studies of reliance on memory, irrelevant personalization, and interactions between user profiles and memory [348, 349, 474] show such failures even when retrieval succeeds. Using a record also depends on authority. EAL-Bench [449] isolates permission errors caused by memory: in synthetic organizational scenarios, it replaces memory while fixing the request, executor, tools, and rules, and uses exact repairs of the memory error and sham edits that leave it unchanged as controls.

Table 17 Representative implementations of M → A. Retained guidance, Action use, and reported comparisons.

Work	Supplied information	Use in Action	Reported evaluation	URL
M → A: personal and task state guide intervention				
User as Code [302]	Typed personal state	Select alerts	Alert-pipeline variants	/GitHub
PASK [468]	Long-term context	Proactive assistance	Short tasks and memory cases	/GitHub
MEMENTO [388]	Personal task memory	Household decisions	Oracle perception and skills	/GitHub
Long-term TOA [315]	Formulated triggers	Follow up when due	Own benchmark tasks	/–
PIS [316]	Typed intentions and status	Act on due intentions	PM-Bench Set-F1	/–
PACE [473]	Personal facts and events	Assess request conflicts	Retrieval/conflict accuracy	/GitHub
EAL-Bench [449]	Memory under fixed rules	Check permission	Memory swaps, fixed authority	/GitHub

5.5 Evaluation

5.5.1 Evaluation Metrics

A single task-success score can hide how an intervention went wrong. Table 18 lists the benchmarks and evaluation settings for action, grouped by the operation they test. Their metrics cover four ways an intervention can go wrong. **Timing** measures separate missed opportunities, premature responses, avoidable delay, and appropriate silence. **Effect** measures separate issuing an operation, completing the work, and benefiting the user. **Authority** checks test each action against the grant in force when it takes effect. **Cost** measures cover execution time, computation, interruption, and recovery effort. Deferred action adds whether a commitment runs when due, runs once, and stops when fulfilled or cancelled. Across interactions, later completion, repeated prompts, correction effort, and changing receptivity reveal effects beyond the immediate response [66, 67, 454]. Privacy complaints and reports of social discomfort provide evidence of harm, but capture only what users notice and report. For all these measures, the length of observation determines whether an evaluation can see the consequence the assistant promises to improve.

Some protocols refine these measures. For spoken interaction, MoshiRAG [227] measures both first-audio-token latency and the delay until key answer content appears, which separates starting a response from supplying useful information. Omni-SafetyBench [475] first probes whether the input was understood, then measures refusal and harmful responses only on understood inputs, and compares this safety across matched text, audio, visual, and combined inputs.

5.5.2 Benchmarks and Evaluation Settings

Executing a request. We treat task suites for requested work [476, 477, 478, 93, 479] as baselines, since each task starts from a request and ends within one session. The benchmarks in the first group of Table 18 differ in the endpoint they test. MCPMark [480] checks resulting application states and repeated success. MCP-Universe [481] combines static grading against fixed references with live-reference evaluators that check against current service data. ClawBench [482] uses live sites but intercepts the annotated final submission, so it can assess the proposed submission without observing its final external effect. HarnessAudit [483] and Claw-Eval [484] use traces and rule checks to find violations that completion scores miss. LifeVal [404] builds requested everyday tasks from its writers' own ChatGPT conversations and memories and credits an action only when the environment's backend state shows that it was done. When the facts that a rubric item needed were moved from the task's context into these past conversations, the tested model met that item 41.5% of the time instead of 55.8%. For spoken tool use, Full-Duplex-Bench-v3 [485] uses real recorded speech and simulated APIs with fixed responses to test whether a correction the user makes mid-sentence reaches the arguments of the tool call that follows. StreamReady [194] scores answers to supplied questions against when the answer evidence becomes available, which tests timing within requested work.

Commitments over time. Requests can also set up work that continues after the session. SMH-Bench [486] tests setting up scheduled home automations, where agents still show weaknesses. JarvisBench [487] lets requested work continue in the background after the user stops watching and tests whether an intermediary

answers questions about that work and asks for the user's judgment at the right moment. π-Bench [445] spreads requests over 20 sessions per user and scores whether agents meet requirements before the user has to state them. Deferred action itself is tested mostly over short horizons, from minutes in Gaia2 [446], which adds deadlines and asynchronous events in stateful apps, to one simulated week in PM-Bench [324], and OptionWorld [488] extends this to 60 simulated days. SentinelBench [238] measures how agents wait for a condition, and IPIBench [129] and ProEvent [335] test changes to commitments during a stream and in ongoing chats. Assistant Benchmark [489] observes real products over several weekdays. In one run, an assistant confirmed a weekday routine and still reported it active, but none of the expected runs arrived. None of these settings tests missed or repeated runs after a restart or a change of permission.

Acting or staying silent. Benchmarks for proactive action pose opportunities that the assistant detects now, with no earlier commitment behind them. Here silence is often the right answer, so a useful benchmark counts not acting as correct when nothing is needed. ProVoice-Bench [490], Pro^{2}Bench [237], ProAgentBench [491], and TriggerBench [325] credit correct silence on cases where no intervention is due. Vinci2's benchmark [283] instead defines allowed intervention periods for each category, counts interventions outside them as errors, and judges content against reference responses. A proposed protocol for unsolicited coding insights [492] would also score when to leave the user undisturbed. PROBE [493] tests unprompted task opportunities, PersonalHomeBench [494] adds a generated household history, and RobotAgentBench [495] asks household robots to find tasks from a scenario and the user's context without an explicit task list. All three always expect an action, so none scores silence.

Benefits and costs across repeated interactions. Across repeated interactions, a benefit can fade, so reported gains should come with their costs and the time horizon over which they hold. HeartSteps [433] ran a six-week micro-randomized trial that repeatedly randomized whether to send each participant a walking suggestion and measured steps in the following 30 minutes. Suggestions increased steps on average, but the effect was largest at the start and diminished over time, which motivates repeated measurement. Exit interviews point to boredom with the messages, but the trial does not test this cause or measure long-term health benefit. Other settings distinguish user outcomes from execution performance: SoCRATES [496] compares mediated and unmediated simulated conflicts through later consensus, while Claw-SWE-Bench [497] holds the model and execution budget fixed and varies the scaffolding.

Action Outcomes Are Delayed and Sparse

Assistance that helps today can become a burden later. Interruptions add up and the effect of repeated suggestions can fade, so immediate acceptance does not show lasting benefit.

Pending work must survive restarts without repeating. Models often repeat a write when a request is still in flight or arrives twice, and no reviewed evaluation tests whether personal commitments are missed or repeated after a restart or a change of permission.

Revising the Action policy needs attribution first. Outcomes arrive late and several interventions can precede one, so the assistant must attribute each outcome to the decision behind it and judge a revision across all of its ongoing responsibilities.

Table 18 Action benchmarks and evaluation settings.

Benchmark / protocol	Evaluation setting	Evaluated content	Framework	URL
Reactive: executing a request				
STRIDE [498]	Continuous video interaction	Response timing	S → A	↗/
AURA [499]	Streaming video questions	Immediate and delayed answers	S → A	↗/
Full-Duplex-Bench-v3 [485]	Real speech, mock tool APIs	Corrections in tool arguments	S → A	↗/
StreamReady [194]	Answer-evidence time windows	Answer correctness and timing	S → A	↗/–
MCPMark [480]	Executable tool workflows	Final states and repeated success	π_A	↗/
MCP-Universe [481]	Live tool services	Static/live-reference grading	π_A	↗/
ST-WebAgentBench [439]	Policy-constrained web tasks	Policy-compliant task completion	π_A, c_t: auth., saf.	↗/
HarnessAudit [483]	Constrained execution traces	Permission and data-flow breaches	π_A, c_t: auth., priv.	↗/
Claw-Eval [484]	Traces, audit logs, snapshots	Completion, safety, robustness	π_A, c_t: auth., avail.	↗/
OpenComputer [500]	Application-state verification	Externally checked task outcomes	π_A	↗/
LifeVal [404]	Personal tasks in simulated sites	Rubric and backend state changes	M → A, $\mathcal{B}$	↗/–
ClawBench [482]	Live sites, submit intercepted	Submission payload and attempt	π_A	↗/
LIMBO [243]	Injected tool-boundary faults	Duplicate effects, overclaimed success	A → S, c_t: unc., avail.	↗/–
HumanCLAW [438]	Simulated humanoid, bystanders	Success, collisions, disturbed objects	A → S, c_t: saf., res.	↗/
NormAct [501]	Simulated social scenes	Goals vs. unstated norms	S → A	↗/–
Deferred: commitments over time				
OmniPro [502]	Synchronized audio and video	Timed replies to instructions	S → A	↗/
IPIBench [129]	Changing standing requests	Trigger timing	S → A, M → A, M → S	↗/
JarvisBench [487]	Background agent work	Status answers and judgment timing	S → A, c_t: res., $\notin \mathcal{R}$	↗/
π-Bench [445]	20 sessions per user	Needs met before being stated	M → A, c_t: unc., $\mathcal{B}$	↗/
PM-Bench [324]	Simulated week of activity	Due, updated, and cross-day tasks	M → A, A → S, $\mathcal{B}$	↗/
SentinelBench [238]	Standing-request monitoring	Completion, reaction time, cost	S → A, A → S, c_t: res.	↗/–
Gaia2 [446]	Stateful apps, simulated time	Asynchronous work and deadlines	S → A, c_t: unc., avail.	↗/
ProEvent [335]	Ongoing chat threads with unannounced changes	Timetable upkeep, noticing cancellations nobody states	S → A, S → M, M → A, $\notin \mathcal{R}$	↗/–
Long-term TOA [315]	Dynamic environment	Follow-up on persistent intents	S → A, M → A, $\mathcal{B}$	↗/–
Assistant Benchmark [489]	Real products, several weekdays	Routines, restraint, permissions	π_A, c_t: auth., $\notin \mathcal{R}$, $\mathcal{B}$	↗/–
OptionWorld [488]	60-day worlds, hidden targets	Risks acted on before deadlines	S → A, M → A, $\notin \mathcal{R}$, $\mathcal{B}$	↗/–
SMH-Bench [486]	Smart-home tasks, including scheduled automations	Grounded control, schedules, clarification, memory	S → A, M → A, c_t: unc.	↗/–
Proactive: acting or staying silent				
Vinci2 [283]	Egocentric intervention windows	Timing and grounded responses	S → A, $\notin \mathcal{R}$, $\mathcal{B}$	↗/
Pro^{2}Bench [237]	Procedural assistance streams	Guidance timing and recovery	S → A, A → S	↗/–
ProVoice-Bench [490]	Voice interaction streams	Intervention choice, accuracy	S → A, $\notin \mathcal{R}$	↗/–
EgoVoice [419]	Egocentric task recordings	Speech timing and relevance	S → A, $\notin \mathcal{R}$	↗/–
ProAgentBench [491]	Real interaction events	Help-seeking timing and intent	S → A, $\notin \mathcal{R}$	↗/–
PROBE [493]	Unprompted task opportunities	Bottlenecks and action choice	S → A, $\notin \mathcal{R}$	↗/–
TriggerBench [325]	Latent constraints	Reminders and false alarms	S → M, M → A, $\notin \mathcal{R}$	↗/–
KnowU-Bench [392]	Emulated phone, simulated user with hidden profile	Act, ask, stay silent, or stop once the user turns down a proposal	S → A, M → A, c_t: auth., $\notin \mathcal{R}$	↗/
PARE [236]	Simulated active phone users	Proposals, acceptance, task success	S → A, c_t: res., $\notin \mathcal{R}$	↗/
PersonalHomeBench [494]	Personalized smart homes	Context-appropriate assistance	S → A, M → A, $\notin \mathcal{R}$	↗/–
RobotAgentBench [495]	Open-ended simulated homes	Task discovery without a task list	S → A, $\notin \mathcal{R}$	↗/–
Authority and consequences across interactions				
HeartSteps [433]	5 daily prompts over 6 weeks	Steps in the next 30 minutes	π_A, c_t: avail., $\notin \mathcal{R}$	↗/–
Oralytics [424]	Sensor-toothbrush clinical trial	Brushing after RL-timed prompts	S → A, $\notin \mathcal{R}$	↗/
DIAMANTE [423]	SMS trial, phone step counts	Steps after learned message timing	S → A, $\notin \mathcal{R}$	↗/–
EAL-Bench [449]	Multi-session permission histories	Memory-induced authority violations	M → A, c_t: auth., $\mathcal{B}$	↗/
PRISK [474]	Profile and memory conditions	Inappropriate personalization	M → A	↗/
PACE [473]	Hidden request conflicts	Conflict retrieval and judgment	M → A	↗/
SoCRATES [496]	Disputes, unmediated baseline	Intervention timing and consensus	S → A, $\notin \mathcal{R}$	↗/
Claw-SWE-Bench [497]	Scaffolding variants	Resolved issues, cost, duration	π_A, c_t: res.	↗/
Efficient Agents [456]	Agent component variants	Accuracy and cost per solved task	π_A, c_t: res.	↗/
AuthMem-Bench [463]	Paired histories that differ only in source authority	Claims upgraded to standing instructions, unauthorized calls	S → M, M → A, c_t: auth., $\mathcal{B}$	↗/–
ATRBench [453]	Learning, then offline sessions	Asking now for later preferences	M → A, c_t: unc., $\mathcal{B}$	↗/
CarryoverBench [503]	One-time injection planted in memory or heartbeat files	Payloads re-firing in later sessions, after the attacker leaves	A → M, M → A, c_t: saf., $\mathcal{B}$	↗/
GAP [241]	Simulated homes over several weeks, LLM-played user	Net user actions saved, and placements the user undoes	S → A, M → A, A → S, $\notin \mathcal{R}$, $\mathcal{B}$	↗/–
MoveCare [158]	Home robot in 25 homes	Reminders, false alarms, follow-ups	M → A, $\notin \mathcal{R}$, $\mathcal{B}$	↗/–
Hyodol [504]	Companion robot, 278 homes	Check-ins and depressive symptoms	π_A, $\notin \mathcal{R}$	↗/–

Framework: π_A marks a protocol that scores the Action policy alone, with no coupling channel; $X \to Y$ denotes a coupling channel; c_t gives the shared conditions that are scored; $\notin \mathcal{R}$ marks a scored decision at a step outside $\mathcal{R}$, where no explicit request is being carried out; $\mathcal{B}$ marks state that is retained when an interaction ends and is used later. A channel is marked only when the score of the protocol depends on that flow between modules. No protocol in this table scores a mechanism revision (θ_t). Condition abbreviations: **priv.** privacy, **auth.** authority, **saf.** safety, **res.** resources, **avail.** availability, **unc.** uncertainty.

6 Applications, System Bottlenecks, and Loop Closure

This section first examines how Sensing, Memory, and Action work together in wearable, digital, and physical applications, and where reported systems meet bottlenecks. We distinguish these applications by the three coordinates of §2.5: interaction domain, participant scope, and outcome horizon. Table 19 records these coordinates for systems that decide for themselves when to act between requests or carry state across interactions, grouped by interaction domain. Within each domain, we compare how systems use current observations, retained context, and action outcomes. Several of these bottlenecks arise because one shared condition, such as a battery budget or a privacy rule, constrains Sensing, Memory, and Action at once. We therefore turn next to shared conditions, and then to the implementation that all three modules share. Closure levels then place every reviewed system on one scale, from retained state through unrequested action to governed revision. Last, we examine how reported systems revise their mechanism. At the highest level, revisions must stay within the shared conditions.

6.1 Applications and Bottlenecks

6.1.1 Wearable Assistance in Daily Activity and Conversation

Wearable assistants use what the user sees and hears to help during everyday activity. They guide a task as it happens or recall it afterwards, and the two need evidence at different times and draw on different amounts of the past. First, guidance needs evidence while the task is under way, so it starts from the current scene, as in Vinci [8] and ProAssist [505]. During a procedural task, an assistant also tracks which step the user is on, and procedural-assistance surveys [507] review how systems recognize steps and detect errors. That estimate must be stable, since a single frame can be misread: Pro^{2}Assist [145] therefore updates its progress record only when successive step predictions agree and does not repeat guidance while the step stays the same. Guidance must also arrive at the right moment, so ProMemAssist [9] decides when help will be timely by modeling the user's working memory, the information the user temporarily holds for the current task. Second, recall draws on longer personal histories [508], which EgoLife [94] provides as daily-life streams for study, and it can wait for better evidence: LightMem-Ego [137] keeps updating current state while a slower consolidation step uses later transcripts to refine its event records.

In conversations and meetings, the assistant keeps listening while it responds, often with several people present, and it makes two decisions. First, it decides when to speak. Full-duplex research studies systems that listen while speaking and examines when they should take or yield a turn [509, 510, 511]. Second, it decides how to deliver the response. For private delivery, LlamaPIE [416] gives short suggestions in the user's ear. SocialMind [246] instead uses an augmented-reality display, which places digital information over the user's view. Sensible Agent [418] chooses from the current context both the content and the modality, the form in which the response reaches the user. A private cue and a spoken suggestion can carry the same content, but the spoken one interrupts everyone present and discloses the content to them. The choice between them depends on whose information the assistant captured, who will receive the response, and which retained context that audience is allowed to see.

Bottlenecks and trade-offs. Wearable assistance runs into three bottlenecks that no single module resolves: the cost of capture, the repair of wrong records, and timing and audience in conversation. First, **the cost of capture** links Sensing to Memory: how much to capture cannot be set from sensing cost alone, because what the assistant captures now limits what Memory can recall later. Resource studies measure sensing and inference costs [182, 122] and streaming efficiency [512, 513, 514], while the other side of the trade-off appears in Memory: more capture preserves more events and adds transmission, storage, and maintenance work, whereas less capture saves resources at the risk of losing personal history. Capture also shares the device budget with delivery and with the checks that other pending tasks need. Second, **repair** links Memory to later interactions with the user: a wrong record has to be found and corrected, by the user or from the assistant's own later observations, because capturing more does not fix it. For example, Memento's [231] multi-day study reports useful contextual recall alongside mismatches in what memories refer to in the scene, and its targeted removal control lets users stop unwanted memories from recurring. User

Table 19 Systems and supporting methods across application scopes. Groups follow the three interaction domains. Each row identifies a core method, a potential bottleneck, and its location in the framework of §2. The application scope of each case gives its interaction domain, participant scope, and outcome horizon (§2.5).

System	Scope	Key method	Bottleneck	Framework	URL
Wearable					
ProAssist [505]	×I×s	Context-triggered step dialogue	Event interpretation	π_S	/GH
ProMemAssist [9]	×I×s	Sensed task state guides help timing	Task-state estimation	π_S	/–
Vinci2 [283]	×I×s	Video memory guides intervention	Compression / selection	π_M, M→A	/GH
VisionClaw [10]	×I/G×s	Visual capture, recall, and actions	Capture budget / privacy	π_S; c_t	/GH
Memento [231]	×I×s	AR recall; referent/removal controls	Recall relevance / timing	π_M, π_A	/–
Sensible Agent [418]	×I×s	Context selects content and modality	Interruptibility uncertainty	π_S, π_A	/–
AiGet [247]	×I×c	Gaze and profile select knowledge	Disruption / repetition	M→S, A→M	/GH
SpeechLess [134]	×I×s	Spatial memory informs intent	Stale / ambiguous state	π_M, M→S	/–
ProAgent [130]	×I×s	Persona priors; feedback updates	Prior error / sensing cost	M→S, π_S	/–
Pro^{2}Assist [145]	×I×s	Motion-based sampling; step checks	Step uncertainty / repeats	π_S, π_A	/–
LightMem-Ego [137]	×I×s	Async. consolidation for recall	Consolidation / recall lag	π_M	/GH
Alpha-Service [131]	×I×–	Event triggers, memory, and tool use	Triggers / authority	π_S, π_A; c_t	/–
SocialMind [246]	×G×s	Persona-conditioned AR suggestions	Audience / context	M→S, M→A	/–
LlamaPIE [416]	×G×s	Audio-guided in-ear suggestions	Turn timing / consent	π_A; c_t	/GH
M3-Agent [95]	×I×s	Entity memory from video and audio	Identity / memory upkeep	S→M, M→A	/GH
Digital					
PASK [468]	×I×s	Intent cues and post-session memory	Intent freshness	π_S	/GH
JoyAI-VL [188]	×I×s	Response gating; asynchronous tasks	Coordination / stale results	π_S, π_A	/GH
PITCH [248]	×I×c	Daily plans; scheduled follow-ups	Plan relevance / receptivity	M→A, π_A	/–
User as Code [302]	×I×s	Typed state; executable alert rules	Schema / state correctness	M space; π_M	/GH
KnowAct [239]	×I×s	GUI skill use; procedure repair	Stale skills / task contracts	θ_t; π_A, A→M	/GH
MOSS [98]	×I×c	Source repair; replay and rollback	Repair transfer	θ_t; π_M	/GH
Ouroboros [99]	×Pub×–	Reviewed code/prompt self-edits	Revision / authority drift	θ_t; c_t	/GH
HealthClaw [303]	×I×l	Health-history updates for later QA	State conflicts / privacy	π_M; c_t	/GH
Oralytics [424]	×I×l	RL prompt policy revised weekly	Delayed brushing outcomes	θ_t; π_A	/GH
Long-term TOA [315]	×I×c	Triggers kept through dormancy	Stale triggers / timing	M→A, A→M	/–
ProAct [152]	×I×s	Idle-time anticipation	Wasted preparation	M→S, π_A	/GH
ProPerSim [369]	×I×l	New version each day from ratings	Simulated users / drift	θ_t; A→M	/GH
Dot† [16]	×I×–	Self-timed wake-ups; delegated tasks	Authority / result checks	A→S, π_A; c_t	/–
Claude Cowork† [410]	×I×–	File access and multi-step execution	Permissions / recovery	π_A; c_t	/–
Muse† [15]	×I×–	Memory, schedules, permission gates	Authority / freshness	π_S, π_A; c_t	/–
Claude Duet† [17]	×I×–	Account checks; approval gates	Urgency / usage limits	π_S, π_A; c_t	/–
Windows Recall† [139]	×I×–	Protected screen-history retrieval	Admission / access control	S→M, π_M; c_t	/–
EAIA† [411]	×I×c	Email triage, drafts held	Authority / edit learning	θ_t; c_t	/GH
Physical					
Hobbit [3]	×I×c	Scheduled reminders; idle patrol	Trigger reliability / alarms	M→A, A→S	/–
STRANDS [4]	×G×l	Visit timing learned from logs	Site-level state / drift	A→M, M→A	/GH
EgoMem [305]	×G×s	Face/voice polling; lifelong memory	Identity / memory upkeep	S→M, M→A	/–
MoveCare [158]	×I×c	Self-timed reminders; result checks	False alarms / missed data	M→A, A→M	/–
PALRO [320]	×I×l	Stored vitals drive reminders	One user / fixed logic	S→M, M→A	/–
PHILIA [5]	×I×s	Remembered preferences guide tasks	Relevance / safe execution	M→A, π_A	/–
AssistantX [2]	×G×s	Cross-domain plans; outcome review	Who is free / verification	A space,A→S	/GH
ABot-Claw [6]	×I×s	Shared visual memory, critic	Progress checks	A→S, M→A	/GH
ABot-AgentOS [290]	×G×c	Cross-session graph, provenance	Identity / commitments	π_M, S→M	/–
PAM [506]	×I×c	Routine memory revised each day	One user / short recordings	M→A, π_M	/–
ElliQ† [319, 362]	×I×–	Context-timed self-initiation	Initiation timing / fatigue	M→A, π_A	/–
Hyodol† [414, 504]	×I×l	Twice-daily voice check-ins	Records shared, not reused	π_M	/–

Framework: M space and A space are the Memory and Action spaces of §2.2, what a module can do and represent; π_S, π_M, π_A are local policies; $X \to Y$ denotes a coupling channel; c_t gives shared conditions. θ_t marks a mechanism revision (§2.1). Scope: interaction domain physical, wearable, digital; participant scope **I** individual, **G** bounded group, **Pub** public; outcome horizon **s** within-session, **c** cross-session, **l** longitudinal, – not reported. † Product.

correction helps only when the user notices the error, so a change that nobody announces must be inferred from the assistant's own later observations.

Third, in conversation, **timing and audience** link all three modules: a response prepared in the background has to be checked against the topic and the audience at the moment it is delivered. Sensing and Memory often finish their work in the background, so a result can reach Action after the topic or audience has changed, while an immediate response can interrupt a useful exchange. More frequent interventions consume attention and leave more outcomes to interpret. Interaction benchmarks examine response timing [515, 516, 419], and ProVoice-Bench [490] and long-term streaming voice tasks [517] also test sustained interaction. Across sessions, a commitment made in a meeting can still matter, but a new audience is not always entitled to its details. The people present have often not agreed to be recorded or served. When Sensing detects that the audience has changed, Memory can restrict what it releases and Action can recheck recipients before delivery, and when the topic changes, Action can rewrite or withhold the queued response.

6.1.2 Digital Workspace and Personal Assistants

Digital assistants carry plans and personal context across applications and interactions, and some also revise the procedures or code used in later work. The digital-colleague perspective [518] connects persistent workspaces, reusable skills, and governance. PITCH [248] schedules morning planning and evening reflection in Slack, and in the evening, it reuses the morning conversation to ask about progress and departures from the user's plan. However, its field-study dialogues include refusals and misplaced follow-ups. Remembering the plan supplies a topic, but the user's current progress and willingness to engage determine whether a follow-up is appropriate. Product descriptions extend the use of retained context to background work, with different choices about when to run and notify the user. Claude Cowork [410] runs delegated workspace tasks on a schedule. Muse [15] combines scheduled and event-driven work with memory across conversations and selective notification, with Sentinel checks for permission-sensitive operations. OpenAI's Dot [16] keeps working between conversations on its own cloud computer, decides when to pause and wake up, and reaches out with results or with decisions that need the user. Claude Duet [17] carries over the memory, skills, and connected apps of the user's Claude account. It sends a brief each morning and checks the connected accounts every few hours, messaging the user only when something needs attention. Windows Recall [139] illustrates the related storage boundary for screen history.

Persistent workspaces also support reusable procedures. ReasoningBank [519] and Agentic Context Engineering [520] retain guidance derived from feedback. Skill-Guided Continuation Distillation [521] and SkillHarness [522] develop reusable GUI procedures. SE-GA [523], SCALE [524], and Darwin Mobile Agent [525] also reuse experience. These methods improve across a stream of separate tasks, the setting of the self-improving agents in §2.5. KnowAct-GUIClaw [239] links reuse to repair: it retrieves stored experience and skills, checks the screen state after each subtask, and repairs a reused skill when it fails before extracting a new one. Its evaluation first adds three components together: a host that manages the user's task, a router that splits requests spanning several apps into subtasks, and experience memory. It then adds skills on top, so each result shows the effect of a combined change. Reusing records and procedures across tasks also requires selecting context for the current audience and applying corrections wherever the old information remains in use.

Bottlenecks and trade-offs. Digital assistance runs into two bottlenecks that no single module resolves: keeping the workspace available and reusing retained context. First, **keeping the workspace available** draws on computation and bandwidth that all three modules share, so background checks and memory upkeep have to be scheduled against response delay across all pending tasks. Efficient Agents [456] compares inference schedules by task success and token cost. DualPath [526] addresses storage bandwidth in KV-cache serving, which reuses stored intermediate results to avoid repeating earlier model computation. FreeToken [527] divides execution between the CPU and the GPU and allocates cache loading on constrained hardware, and measurements on local devices [528] report the energy and latency of each model call. These studies measure the cost of calling models and loading context for them. Background checks by Sensing and maintenance by Memory compete with user-facing Action for this capacity, so more frequent checks can raise costs and delay responses. Claude Duet [17] states this trade-off directly: it checks connected accounts only every few hours,

because continuous checking would use up the user's plan limits much faster, and its background checks count toward the same usage limits as the user's other work with Claude. The competition grows with the number of pending tasks, since each unresolved task needs checks of its own.

Second, **reuse** links Memory to Action: when reused context fails, repair requires telling apart missing evidence, which Sensing must acquire, wrong retained state, which Memory must correct, and valid state used outside its current scope, which Action must not apply. Reuse can hurt when the interface, task, or permission has changed. For example, PAST-Bench [24] reports task-stream cases where removing a persistence component improves performance. Permissions and inputs can also change without notice, so an earlier grant can lapse before it is used, and content read without a request cannot be trusted by default. An external task record can recover a missed completion event, while a user correction can expose an obsolete assumption.

6.1.3 Physical Assistance and Collaboration

Physical assistance requires reliable task execution and coordination with people, and a few deployed systems already sustain that assistance between requests. They sustain it in different ways: ElliQ [319, 362] through remembered preferences, Hobbit [3] through execution verification, and STRANDS [4] through outcome-driven scheduling. ElliQ initiates conversations several times a day, using ambient sensing, the user's availability, and remembered interests to decide when to speak and what to raise. Across multi-week deployments in private homes, Hobbit pairs scheduled reminders with background fall detection and a patrol after long inactivity. It checks each grasp before placing the object on its tray and tries once more if the grasp failed. STRANDS schedules visits across days in a care facility and keeps state that serves the residents as a group. It uses logged interaction outcomes to adjust when and where it goes next. Two home-care studies bring the same continuity to health checks. MoveCare [158] times reminders in older adults' homes and closes each one when the requested measurement arrives. PALRO [320] stored one older adult's vital signs over 334 days and reminded the user whenever a daily measurement was missing, thanking the user once it arrived.

PAM [506] studies routine-conditioned initiative directly. A visual task predictor tracks which phase of a routine the user is in, using a memory of the activity's phases and its assistance rules, and selects a task or stays silent. At the end of each day, a separate reflector reviews the day's observations and actions and revises that memory. Its five-day prediction study uses short recordings of one user's desk routine. This supports a bounded test of memory-guided assistance and daily revision, not of generalization to unseen users or long-term autonomous deployment.

Recent robot agent systems also carry personal state across tasks, but they mostly act on instruction. PHILIA [5] keeps long-horizon memory of people and preferences across several robots. ABot-Claw [6] uses a critic to correct and replan during execution, and it shares visual memory across embodiments, the physical robot bodies in which the agent operates. Other systems organize personal state across interactions. ABot-AgentOS [290] stores identities, preferences, and social commitments in a cross-session graph, a structure that represents entities and their relationships. AssistantX [2] likewise keeps a graph of people, places, and items, with ownership and who is available, through a month-and-a-half office deployment. To associate memory with returning users, EgoMem [305] recognizes them by face and voice and updates their memory after each dialog. Every evaluated episode in these systems starts from an instruction or a live exchange. Revisiting an unresolved commitment without a new request needs the scheduling and monitoring that the deployed systems above provide.

Bottlenecks and trade-offs. Mistakes usually cost more in physical assistance. Physical actions add risks of contact, collision, and movement, and a wrong action can hurt the user or people nearby. In digital and physical assistance alike, whether an action can be undone depends on its effect. Physical assistance runs into two bottlenecks that no single module resolves: checking results and judging whether the assistance was useful. First, **checking results** links Action back to Sensing: how much to check depends on the consequence of an error and the cost of delaying or repeating the action. Reviewing the scene helps resolve uncertain execution, but extra observation and replanning take time and resources. Because the assistant often acts while the user is away, a physical effect that cannot be undone, such as a collision, can occur before anyone is able to stop it, and people who have not agreed to the assistant share its consequences. Second, **judging**

whether the assistance was useful links Action to Memory: the assistant can tell whether it helped only if Memory keeps the user's response and later task progress.

6.2 Shared Conditions and Implementation Across Modules

Many of the bottlenecks above come from what the three modules share. In a running assistant, Sensing, Memory, and Action work at the same time, under the same conditions and on the same implementation. In this section, we examine shared conditions that constrain several modules at once. Such a condition holds only when every module it constrains meets it, and a limit or failure in one module carries over to the others, because information flows between them and they often share a device, a model, and permissions. We first take these shared conditions in turn and then the implementation that all three modules run on.

6.2.1 Shared Conditions

Authority. A permission limits what the assistant is allowed to read as well as what it is allowed to change. Muse [15] separates read and write permissions where a connected service supports it, so a grant for the calendar can allow reading without allowing edits. With this grant, the assistant can see when an event ends but cannot move it. One grant therefore sets the boundary for two modules: what Sensing is allowed to obtain and what Action is allowed to change. Claude Duet [17] draws this boundary by the kind of effect: it reads, drafts, and prepares work on its own, but it shows exactly what it is about to send, pay, sign, or delete and waits for the user's approval. The grant must also hold at the moment the assistant acts, and Dot [16] checks it then, reviewing each action that could affect the user's accounts or share information against the user's permissions and custom rules. The review lets the action proceed, asks for approval, or hands the step to the user. Assistant Benchmark [489] tests such grants directly, giving an assistant read-only access and one standing rule never to send email or spend money without asking. In one of its logged runs, Poke kept drafting email replies after repeated instructions to stop. A grant can also limit revisions to the assistant, for example allowing changes to how it words its messages but not to which actions it takes without asking.

Privacy. A privacy requirement can reach further and bind all three modules. In the confidential meeting of §2.4, Sensing is allowed to process other people's speech, but Memory is not allowed to keep it and Action is not allowed to pass it on. In a group chat, for example, the assistant can stay silent and send a follow-up privately to the one person who has to act. However, no reported system controls all three. LlamaPIE [416] and SocialMind [246] control only Action by keeping delivery discreet, the first in the ear and the second on an augmented-reality display. Windows Recall [139] controls what Sensing captures and what Memory keeps but not what Action passes on, yet an always-on assistant in that meeting needs all of these controls at the same time.

Safety. Safety concerns harm that the assistant's actions can cause, yet it also binds Sensing and Memory. Before a home robot moves its arm, Sensing supplies recent evidence that no one is within reach, and an older observation in Memory cannot stand in for it. Memory can also carry a risk forward, since a procedure that once saved time by skipping a check can repeat that shortcut in every later run. However, reported work addresses these parts separately. On the Sensing side, ICWM [529] probes a robot's camera and control configuration with motions chosen to avoid contact with task objects. On the Action side, AGrail [448] adapts safety checks to each agent task. On the Memory side, studies of self-improving agents report retained memory and skills that become unsafe as the agents improve themselves [530, 531]. No single control covers the path from observation through retained procedures to the action [532].

Resources. Authority, privacy, and safety limit what the modules are allowed to do, whereas resources and availability limit what they are able to do. Resources are shared most directly, since every module draws on the same battery, computation, storage, and user attention, and pending commitments compete for them. User attention shows this sharing most clearly, because each module can ask the user for something. Sensing asks when its estimate is uncertain, as PrISM [135] asks about the user's current step. Memory can ask before it keeps a record, as SpeechLess [134] confirms each low-confidence interpretation before it enters memory. Action asks the user to confirm an intervention before it takes effect, and each reminder it

delivers also spends attention. The user answers all of these questions with the same attention, so a question from Sensing leaves less room for a reminder from Action. One resource can also be spent to save another: Pally [312] checks whether the user has already handled a task before sending a reminder, spending a check to avoid an unneeded interruption. Storage also links Memory to Action, since an action record deleted to save space can no longer connect a later outcome to the decision it concerns.

Availability and uncertainty. Availability constrains Sensing, Memory, and Action together, since a decision requires all the information and interfaces it relies on to be reachable at that moment. When an account is revoked or a service goes down, Sensing cannot read the current state, Memory cannot reach the records kept there, and Action cannot complete or confirm its step. A fallback helps only if it does not depend on the same account, permission, or service as the check it backs up, so a recovery plan needs to know which components share such dependencies. Externalization moves memory and skills out of the model into separate stores and tools [533]. It adds components of this kind and makes these dependencies more important to trace. Uncertainty carries across modules in the same way. An estimate that Sensing marks as uncertain stays marked only if Memory writes its confidence with the record, and otherwise Action treats a stored guess as a fact.

6.2.2 Implementation: Organizing Context Across Requests

What operating between requests asks of the implementation. The implementation consists of the foundation model and its scaffolding, and it must organize model calls and their context across requests. An assistant built for a single task starts when the user makes a request and stops when the task ends. Everything it needs can stay in the model's context until then. An always-on assistant can rely neither on requests to start each step nor on one context lasting until the work ends. For example, an assistant that has promised to rebook a delayed flight must wake when the airline posts a new time, with no new request. If that happens days later, its permission to use the booking account may need a new check. If its rebooking rule was revised in the meantime, a late complaint must be traceable to the rule that was in effect. Below, we take the model, prompts and context construction, storage, tools and APIs, and the harness in turn and describe how reported systems meet these needs across modules and requests. Storage keeps the pending commitment, tools recheck the permission, and the harness wakes the assistant and records which rule each run used. Choices that concern a single module are left to the section on that module.

Model. One foundation model often carries several modules, so its parameters are shared by the whole assistant. A change to them reaches every module the model serves, not only the one whose feedback prompted it. A model's weights are the numerical parameters that determine its behavior, and assistants that revise them also choose when a new version starts to serve. ProPerSim [369] starts a new version each day, and OpenClaw-RL [240] publishes revised weights at controlled points. Between these points the model stays fixed, and whatever changes from one request to the next reaches it through its context.

An always-on assistant also needs forecasts of two kinds: whether the user will welcome help now, and how the environment will respond to an action, for example whether a robot's motion is safe. A predictive or world model can supply both [32], and it is shared in the same way as the foundation model, so a wrong forecast can mislead several modules at once. Because the user's habits change, such a model also has to keep adapting. Time2Stop [370] retrains its overuse predictor from the user's responses to its interventions, and ICWM [529] adapts to a new robot configuration through its context alone, without updating parameters.

Prompts and context construction. When the implementation calls a model that keeps nothing between calls, the scaffolding must assemble each call's context from stored records, current observations, and the task at hand. One record can enter the context of several modules. In AiGet [247], for example, the same user profile enters both the perception prompt and the generation prompt, so prompt contents carry $\mathrm{M} \to \mathrm{S}$ and $\mathrm{M} \to \mathrm{A}$. In these implementations, a channel delivers information into a call's context, and HALO locates channels and changes their signals at these interfaces (§7). The context is also limited in size, and a long-running assistant decides what to bring back into each call. MemGPT [19] evicts older messages from its context but keeps them retrievable in recall storage.

Streaming implementations can instead carry dialogue history through ongoing inference. DuplexOmni [534] and MOSS-Live [206] process audio and video and generate speech in an interaction layer, while a separate thinking layer performs reasoning and tool use asynchronously. Returned results enter the ongoing interaction, and the interaction layer can stop background output when the task changes. These components therefore update their context at different times. Keeping this session state supports continuity within an interaction, while state durability requires relevant state to survive interaction boundaries.

Storage. Records that outlast a call are kept outside the model's context, in stores that survive both the end of an interaction and the idle time between requests. AssistantX [2] keeps a long-term graph while it clears short-term memory after each instruction. Long-term TOA [315] keeps a reminder tracker through dormant periods. User as Code [302] saves facts into a typed user project, whose fields specify the kinds of information stored. Several modules read the same store, as the AiGet profile shows. A store that is lost or out of date therefore misleads all of them at once and is one of the shared dependencies that a recovery plan tracks.

Tools and APIs. Tools and APIs connect the assistant to calendars, messages, files, devices, and outside services, and one tool can serve several modules (§2.4). For example, one mail connection lets Sensing read new messages, Memory search earlier threads, and Action send replies. In current products, the user sets up each connection, usually by signing in to the service the first time a tool needs it [298, 410, 412, 535]. The assistant then works within the access that this account already has [410, 16], so outside services record its actions as the user's own. Many of these connections now follow the Model Context Protocol (MCP), an open standard for connecting assistants to the systems that hold data [536]. These products all support it [298, 410, 536], and the protocol leaves the choice of servers and tools, and the way consent is requested, to the host application [536].[1]

Once set up, these connections stay available across conversations, so runs that no request starts can use them. ChatGPT tasks can start when a new Gmail message, Slack message, or GitHub pull request arrives [298], and Gemini and Claude run scheduled tasks over connected apps [412, 410].[2] Such runs also need somewhere to execute after the user closes the app. Gemini Spark and Claude Duet run on dedicated cloud virtual machines, so the user's laptop or phone need not stay on [412, 17].[3]

Access to each tool can still change while the assistant waits, so reported systems confirm authority at the moment of use before tools with outside effects. VisionClaw [10], for example, confirms the recipient and content before it sends a message. A security audit of OpenClaw draws the same lesson for long tasks: permissions have to follow a request for as long as the agent works on it, and turning access off has to reach work that is already running [332].[4] Withdrawing a connection, however, does not undo earlier work. Stopping or deleting a Dot leaves its completed actions in place [16],[5] and information that Muse already used can remain in its memories [15].[6]

Connected accounts also bring in content that nobody has checked. An email or calendar invitation can carry instructions that the assistant follows later, and researchers have used such content to make production assistants leak data and control smart-home devices [537, 538]. A tool server can also hide instructions in its tool descriptions, an attack called tool poisoning. In MCPTox, such attacks make the agent follow the hidden instructions in 36.5% of attempts on average across 20 agents [539]. Proposed defenses limit what each tool is allowed to do [540, 541, 542, 543].

Isolating the computer the agent works on addresses a different failure. It keeps code that the agent runs from reaching beyond its environment, but it does not stop the agent from misusing an interface it has been granted, such as a connected mail account, so escape from the environment and misuse of a granted interface need separate checks. Muse runs the agent in a container on a dedicated virtual machine for each user and keeps its credential and permission services outside the container [15]. Dot's cloud computer is kept separate

[1] MCP specification: https://modelcontextprotocol.io/specification/
[2] Gemini schedules: https://support.google.com/gemini/answer/17094710
[3] Gemini Spark: https://blog.google/innovation-and-ai/sundar-pichai-io-2026/
[4] OpenClaw security audit: https://openclaw.ai/blog/openclaw-trail-of-bits-engagement-recap
[5] Dot controls: https://learn.chatgpt.com/docs/dots/controls
[6] Muse connectors: https://www.meta.com/help/artificial-intelligence/1687253048996149/

from the user's own devices and their signed-in browser sessions [16],[1] and Manus gives each task its own isolated cloud virtual machine [408]. Isolation can also be partial or off. All of a user's Grok Bots share one cloud computer with its files and browser sessions, and the documentation warns against treating separate Bots as a security boundary [242].[2] OpenClaw's sandbox is off by default, and its Gateway, which holds channel connections and credentials, always runs on the host [332].[3]

Harness. The harness runs the loop: it decides when a step happens, what runs in it, and which version of the mechanism is in effect. First, between requests it creates the steps that *operational continuity* requires. Muse [15] runs timed and event-driven work and notifies the user only selectively. In Long-term TOA a monitor wakes the agent, which then follows up or waits, and MoveCare [158] runs a weekly check that sends a weight reminder only when none was recorded. When several tasks are pending at once, the harness also decides where each new event is handled. DeepSeek's Super-agent [544] runs each task it has taken on in its own session, so that the details of different tasks do not mix. One persistent session talks with the user and assigns new work. Second, the harness can hold an operation until the user answers. Muse [15] does so for permission-sensitive operations through Sentinel—the host-side agent that acts as the final permission for every outside action. Finally, the harness records which mechanism version each run used, so that a late complaint can be traced to the rule then in effect. PILOT [545] starts new workers on the revised scaffolding, and Ouroboros [99] keeps its live and benchmark histories apart. In §6.4, we examine how such versions are accepted and activated.

6.3 Levels of Loop Closure

To bring together the applications of §6.1 across the three domains, we divide assistants into six levels by how completely their modules close the always-on loop, following the definition in §2.1. These levels are cumulative: L0 is a session-local baseline that keeps nothing across interactions, L1 adds state durability, L2 lets the retained state guide decisions, and L3 adds operational continuity. From L3 on, both properties hold, so L3 is the first always-on level. L4 adds attribution: the assistant links outcomes to the decisions behind them, so a revision can rest on evidence. L5 adds governed self-evolution: the assistant revises its own mechanism, which extends the loop, and every revision stays within the shared conditions. Figure 10 summarizes the levels, and Table 20 lists what each one requires.

We place each system at the highest level for which every prerequisite holds. Evaluating systems by loop completeness is necessary because a capability behaves differently once the full always-on loop is running. Acting without a request requires durable state to stay personal over time, and revising a mechanism is far harder when actions happen between requests and outcomes arrive late. Consequently, a system with a strong late-stage capability still sits at a low level when it does not yet meet the requirements for an always-on assistant, and Table 3 names such kinds of system. LlamaPIE [416], for example, decides on its own when to whisper a suggestion, the kind of decision that L3 adds. It keeps nothing after the conversation, however, so it is a streaming assistant without durable user state and stays at L0. VisualClaw [185] likewise reports all five evidence steps of self-evolution but acts only when asked, so it is not always-on and stays at L2 as a self-improving agent.

L1: Retained state. At L1, the assistant carries information across sessions. Compared with L0, it adds a memory that one session writes and a later session can read. Two reviewed systems stop at L1. MOSS [98] maintains user state across sessions without using it to personalize a later decision, while Hyodol [414, 504] shares its twice-daily check-in records with caregivers but is not shown to use them later.

L2: State-guided decision. An L2 assistant lets its stored records change later decisions about what to sense or how to act. Compared with L1, it adds a Sensing or Action policy that reads memory together with the current context, but the assistant still acts only as requested. Research systems at L2 mostly study how memory is built and updated. Egocentric assistants observe the world from the user's viewpoint, and they

[1] Dot computers: https://learn.chatgpt.com/docs/dots
[2] Grok Bot security: https://docs.x.ai/grok-bot/approvals-security-and-privacy
[3] OpenClaw sandboxing: https://docs.openclaw.ai/gateway/sandboxing

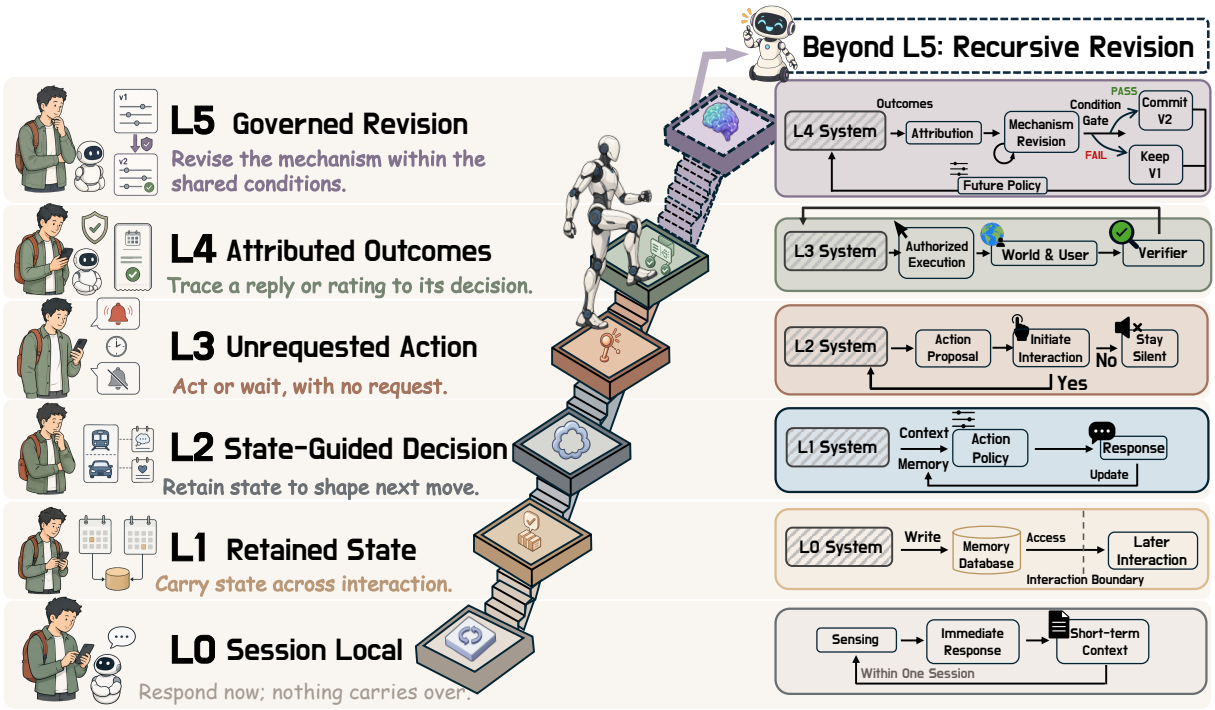


Figure 10 Levels of loop closure. Each level keeps the capabilities of the level below and adds one more. L0 responds within one session and carries nothing over, L1 adds state retained across interactions, and L2 adds state-guided Sensing or Action. L3 adds decisions to act or wait without a request, which makes the assistant always-on. L4 adds links from outcomes to their decisions, and L5 adds revision of the mechanism within the shared conditions.

include VisionClaw [10], M3-Agent [95], and Vinci [8]. They answer later queries from retained observations, while HealthClaw [303] conditions an action on stored user records. Products reach L2 in the same way, but they draw their records mostly from all-day capture or from accounts the user has connected. The RayNeo iO [546] glasses answer later questions from an all-day audio record, Oura Advisor [547] draws on the ring's continuous health data, and Gemini's Daily Brief [548] reads the user's connected Gmail and Calendar.

Two kinds of assistant look always-on but stay at L2. Background capture, as in LightMem-Ego [137] and Windows Recall [139], adds state between requests while Action waits for the user to ask. Scheduled assistants act at every scheduled run instead of choosing whether to act. CareCall [311] calls its users every week, Claude Cowork and Manus [408] run the tasks a user has scheduled with retained context, and the Daily Brief, Oura Advisor's check-ins, and the Today [549] assistant arrive at fixed times. Some research systems at L2 also check outcomes or revise their own rules, which can make them look like L4 or L5, but they stay at L2 because they do not decide whether to act between requests. PAHF [232], the method in Pro^{2}Bench [237], PHILIA [5], and ABot-Claw [6] correct the task in progress. PITCH [248], Bloom [550], and MindShift [551] instead link replies or ratings to messages they send at every scheduled time or trigger. Revisions can also persist at L2: VisualClaw [185] adds skill cards, written instructions for performing tasks, after failures so that later steps can use them.

L3: Unrequested action. L3 is the first always-on level, because the assistant begins to decide for itself whether or when to act without a request, and waiting is one of its choices. Compared with L2, it adds this decision between requests: the assistant proposes an action, then either initiates the interaction or stays silent. Acting as asked at a time, event, or condition that the user fixed carries out an earlier request and stays at L2. The test is who decides whether a run reaches the user. If the user fixes when an automation runs but the assistant judges whether the result is worth sending, the assistant makes that decision. Research

systems such as PASK [468], ProAct [152], SocialMind [246], and User as Code [302] act when a monitored signal crosses a threshold that the assistant, not the user, sets. User as Code, for example, writes its own alert rules from facts the user mentions and keeps them for later sessions. They measure these decisions in benchmarks, simulations, or short user studies.

Products also decide when to reach the user, but where the research systems test explicit trigger rules in controlled settings, products build this decision into everyday workflows. Muse [15] wakes on a timer or an event and notifies the user only selectively. Brea [409] likewise runs automations on their own schedules or events but messages the user only when a run finds something new, and Orilo [552] watches the topics that a user names and lets the user know only when something important happens. Claude Duet [17] flagged, without being asked, a property showing that sat on a tester's calendar for the wrong day, although its announcement notes that it can still misjudge what is urgent. Phone systems make the same choice inside ordinary apps. Pixel's Magic Cue [553] offers information from other apps while the user replies to a message and shows nothing when it finds no relevant suggestion. In the home, aibo [554] chooses how to behave between commands. Several products are placed only from announcements or third-party tests, as when Orchid [313] followed up unprompted on fares two days later and Instinct [555] flagged a duplicate rental booking in Assistant Benchmark [489] runs. Many of these products decide through a relevance filter that stays silent by default to limit interruptions. None reports how well that filter decides, and none is shown to link an outcome to its decision and use it later. Meta states that Muse [15] learns over time which decisions need the user's sign-off and that it builds its own tools and edits itself.[1] However, no reviewed source links these changes to the outcomes of earlier decisions, so Muse stays at L3.

L4: Attributed outcomes. L4 attributes outcome evidence to the decision behind it, so that the assistant can check or correct its assistance. The evidence arrives after the decision, from the world or the user. Compared with L3, L4 adds this attribution, either through an authorized execution whose effect on the world and the user is checked or through the user's reply to a reminder. The link is easiest to make when the outcome names its decision and hardest when the outcome arrives late or only implicitly, and we order the reported evidence by this difficulty. Most reported links come from responses that name the decision, such as a rating, removal, dismissal, or comment tied to the suggestion that caused it. ProPerSim [369] ties each rating to the recommendation that prompted it. Memento [231] responds to removal by stopping the removed memory from resurfacing. Time2Stop [370] uses agreement or disagreement with each intervention to retrain its overuse predictor, a model that estimates excessive use. ProAgent [130] stores each suggestion it delivers with the sensory context and the user's verbal feedback, and infers from them when similar suggestions are unwanted.

Products and other assistants extend this attribution from explicit dismissals to later responses, checks of their own execution, and delayed outcomes. Leon [556] stops repeating a proposal its owner declined, and Town [557] and Xiaoyi Suggestions [421] suggest less of what the user dismissed. A later response can also close a reminder or prompt. MoveCare [158] logs a call for help as a false alarm when the user declines to confirm it, and alerts caregivers to a weight change only after a repeat measurement confirms it. Pally [312] closes a bill reminder once a payment confirmation arrives. Other assistants check their own execution. Hobbit [3] and Lio [558] verify each grasp with an immediate check through A$\rightarrow$S and retry a failed one, and Dot [16] checks the results of the tasks it delegates. Fewer assistants link a delayed or implicit outcome to the decision behind it, which is the harder case. STRANDS [4] adjusts later visits from logged outcomes. Some mobile-health interventions learn online by updating their policies as observations arrive during use. Examples include HeartSteps [559], Oralytics [424], MiWaves [425], and DIAMANTE [423], which decide at set times whether to send a prompt and match later activity or self-reports to that decision. Unlike the products, these trials also randomize the decision to test whether it changed the later outcome.

L5: Governed revision. An L5 assistant revises its mechanism only within the shared conditions c_t, the limits set by the user and the world. A control applies these conditions to every revision before it takes effect, so that no revision can change these limits, and the revisions that pass carry over into later runs. Compared with L4, where several assistants already revise their model or policy, L5 adds this control over the revision itself. The control takes either of two forms: the user's approval of each change or limits set in advance on

[1] Muse approvals: https://www.meta.com/help/artificial-intelligence/1385290430137537/

what automatic revision is allowed to change. The two forms differ in cost: approving each change keeps the user in the loop, whereas limits set in advance let the loop keep revising without the user's attention. Only five reviewed works reach L5, all of them research systems, and each only under a stated condition. Ouroboros [99] meets it in its advanced mode: governance files are protected from ordinary writes, and a blocking review checks each change. The spending limit and the stop command also sit outside the code it is allowed to change. Enoch [560, 447] meets it when its human custodian merges every reviewed change. The protections are procedural, however, and the tests use mocked models, substitutes that simulate model responses during testing. HeartSteps, Oralytics, and MiWaves meet it because each revision changes only the parameters of a policy whose form stays fixed. The probability of sending a prompt has fixed bounds that no update can change, and these bounds limit how much of the user's attention the prompts can take. HeartSteps also never prompts a participant who may be driving. In all five, the control either narrows what a revision can change or places a review before each change takes effect. By contrast, the assistants that stay at L4 show how such a control fails when it lies within reach of revision. Hermes Agent [333] writes skills freely by default. Even with write approval on, the agent can still overwrite the configuration files through its shell, the interface for running system commands. OpenClaw [332] stays at L4 even in its review mode, because it records lessons in the file that lists which actions need the user's approval, so a revision can rewrite that list. ProPerSim and EAIA [411] revise their model or prompts from feedback with no reported control that applies the shared conditions to the revision. Self-evolution can therefore appear below L5, but what a revision rests on differs by level. At L3, as in User as Code, a revision is not traced to the outcomes of the assistant's own decisions. At L4, as in these assistants, it can rest on such outcomes but is not governed. Only governed self-evolution defines a level of its own, L5. In §6.4, we examine these revisions in detail.

Table 20 What each loop-closure level adds. Levels are cumulative, and each mini loop shows the modules and the channels a level requires, in the style of Table 2. L0 lacks *state durability*. L1 adds retention in Memory, which its mini loop draws as S→M. L2 lets retained state guide decisions through at least one of the two edges from Memory, but like L1 it acts only as requested. L3 adds no new channel but decides between requests whether to act, so from L3 on both properties of the continuing loop hold, and the shaded header over L3–L5 marks these always-on levels. L4 adds attribution, which needs at least one of the two edges from Action. A check made during execution through A→S suffices, and A→M is needed when the outcome arrives later. L5 further requires governed self-evolution. ✓ marks what a level requires, and an empty cell means not required.

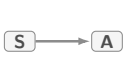
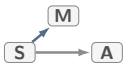
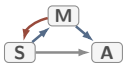
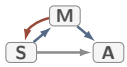
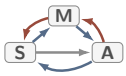
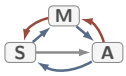
Level	L0 Session local	L1 Retained state	L2 State-guided decision	Always-on L3 Unrequested action	Always-on L4 Attributed outcomes	Always-on L5 Governed revision
Modules and channels						
Continuing loop						
State durability		✓	✓	✓	✓	✓
Operational continuity				✓	✓	✓
Attribution					✓	✓
Governed self-evolution						✓

Table 21 applies these criteria to the reviewed systems, and some entries cover specific variants of a system. The variants differ in backend or operating mode. VisualClaw refers to its Claude Code backend. Ouroboros lets the owner choose a runtime mode, and the paper does not say which mode its long-running public deployment Hope used. The Ouroboros entry therefore denotes the advanced mode, which lets the agent revise its code while the governance files stay protected from ordinary writes. MemGPT and MemoryOS [284] are placed by their chat evaluations, which leave out MemGPT's general support for timed events. MICoBot [407] retains constraints only within one trial. Bee [310] is placed by its Actions instead of its scheduled Daily Insights, and Xiaoyi Suggestions by Huawei's suggestion cards instead of the assistant's scheduled tasks. EgoMem is placed by the architecture its paper describes, whose proactive initiation is not separately evaluated. Because placements rest on what sources report, the level given to each system is a lower bound. HALO (§7) reviews the channel implementations of 40 works, including the five channels beyond the basic S→A path.

Table 21 Reviewed systems by level of closure. Rows follow the three interaction domains, and within each domain we separate research systems from products, including open-source releases. Each system is placed at the highest level that its reported evidence supports, which makes the placement a lower bound. Products are placed from documentation, released code, announcements, field reports, or third-party test runs, not from controlled evaluations. Levels from L3 onward count as always-on, while a system that acts proactively within one session but keeps nothing across sessions stays at L0.

L0 Session Local	**L1** Retained State	**L2** State-Guided Decision	**L3** Unrequested Action **(always-on)**	**L4** Attributed Outcomes	**L5** Governed Revision
Wearable					
ContextAgent [96] Sensible Agent [418] Pro^{2}Assist [145] ProAssist [505] ProMemAssist [9] LlamaPIE [416]	—	VisionClaw [10] SpeechLess [134] M3-Agent [95] LightMem-Ego [137] Egocentric Co-Pilot [151] Pro^{2}Bench [237] Vinci [8] EgoLife [94] IPIBench [129]	Vinci2 [283] AiGet [247] Alpha-Service [131] SocialMind [246]	ProAgent [130] Memento [231]	HeartSteps [559]§
Products					
—	—	Oura Advisor [547] RayNeo iO [546] Even G2 [561] Memoket Gem [562] Soulog Sense [563] MemoMind One [564] SwitchBot AI MindClip [565] Forth [323] Oak1 [566] Ona [322]	Omi [138] Bee [310] Looki L1 [567] Smart Stack hints [568]	—	—
Digital					
AURA [499] JoyAI-VL [188]	MOSS [98]	VisualClaw [185] PAHF [232] KnowAct-GUIClaw [239] HealthClaw [303] MemoryBank [18] MemGPT [19] MemoryOS [284] CareCall [311] PITCH [248] Bloom [550] MindShift [551]	PASK [468] Proactive Agent [12] AwareLLM [14] ProAct [152] User as Code [302]	Long-term TOA [315] ProPerSim [369] Time2Stop [370] DIAMANTE [423]	Ouroboros [99]§ Enoch [560]§ Oralytics [424]§ MiWaves [425]§
Products					
—	—	Windows Recall [139] Claude Cowork [410] Manus [408] Gemini Daily Brief [548] Today [549] Hark Pro [569] Sai [570]	Muse [15] Brea [409] Orilo [552] Claude Duet [17] Orchid [313] Pixel Magic Cue [553] Now Nudge [571] Xiao V Suggestions [572] OPPO Xiaobu [573] Instinct [555] Kindroid [574] LIA [575] SPECS Intelligence [576] Violoop [577]	OpenClaw [332] Hermes Agent [333] EAIA [411] Leon [556] Inbox Zero [314] Dot [16] Pally [312] Town [557] Catch [578] Folk [358] Grok Bot [242] Caddy [351] Xiaoyi Suggestions [421] HeyClicky [579] Vellum [580]	—
Physical					
MICoBot [407]	—	AssistantX [2] ABot-AgentOS [290] PHILIA [5] ABot-Claw [6]	EgoMem [305] PAM [506]	Hobbit [3] STRANDS [4] MoveCare [158] GAP [241] PALRO [320]	—
Products					
—	Hyodol [414]	—	ElliQ [319] Amazon Astro [581] aibo [554] Alexa+ [582]	Lio [558] Moflin [583] Loona Deskmate [584]	—

§ Meets L5 only under a stated condition.
Ouroboros: achieves L5 in its advanced mode, where a blocking review checks each change and the spending limit and stop command sit outside the code it may change.
Enoch: achieves L5 when its human custodian merges every reviewed change.
HeartSteps, Oralytics, and MiWaves: achieve L5 because each revision changes only the parameters of a fixed-form policy.

6.4 Self-Evolution in Deployed Loops

This section turns from the continuing loop to its self-evolving extension. Here REVISE (Eq. (3)) changes the mechanism while the assistant keeps serving the same user, and the mechanism is the implementation described in §6.2.2. We first place each reported revision in the framework. A revision changes the space or the policy of a module, depends on the channels that carry its evidence and can change what they carry, is made through a component of the implementation, and may or may not be governed. We then follow the revisions by component and ask of each what evidence prompted it, how it was accepted, and when it took effect, which follow the five steps of evidence in §2.1 from record to later use.

Why revision is harder inside an always-on assistant. Most work reviewed in surveys of self-evolving agents [42, 55, 43, 56] runs in training pipelines or software repositories, where a candidate change is cheap to test. The revisions target different components: model weights [585], skill memories [586, 587], program code [588], or the harness around the model [589, 590, 591]. Other methods change how improvement proceeds by letting the model choose its optimization procedure, the method used to search for improvements [592]. Robot policies improve over rounds of collected experience [593, 594, 595, 596]. A continuing personal assistant instead learns from operating experience, which adds four difficulties. First, records and outcomes come from the assistant itself, so what was observed, kept, and done limits what can be learned. If the reminder assistant, for example, never observed whether the report was sent, its records cannot recover that fact. Second, a revision changes which outcomes are recorded later: a new rule that checks the calendar less often also changes what the assistant sees afterwards. Third, attribution is hard because feedback from one user is sparse, delayed, and open to competing explanations. Fourth, later use begins while commitments made under the old rule are still pending, and the user expects them to be kept as promised.

What a revision changes. A revision can widen or narrow a module's space, the options it has, or change its policy, how it chooses among them. In Action, VisualClaw [185] adds skill cards after failures, SkillSmith [597] revises skills together with the tools they call, and PANDO [288] retires skills that keep failing, which changes the options, whereas ProPerSim [369], Time2Stop [370], and the mobile-health trials [559, 424, 425] change whether and when to intervene. In Memory, AdaMem [299] revises its write policy and EvolveMem [598] its retrieval rules. No reviewed system revises Sensing, either when it observes or what wakes it. The channels explain much of this pattern. The evidence for a revision arrives through $\mathrm{S}\to\mathrm{M}$ and $\mathrm{A}\to\mathrm{M}$, as feedback on what Action delivered and as records of what it did, whereas a missed observation sends nothing through any channel, so Sensing rarely receives a signal to revise against. A revision also changes what flows to other modules: when EvolveMem revises its retrieval rules, it changes what Memory supplies to Action ($\mathrm{M}\to\mathrm{A}$), and a rule that sends fewer reminders also changes what Memory records about them ($\mathrm{A}\to\mathrm{M}$). A change of space matters most between requests, since a new skill or tool can reach outside effects that the user never reviewed.

Revising model parameters from personal feedback. Because a revision of shared weights reaches every module the model serves, feedback on one kind of decision cannot show whether the others got worse. ProPerSim [369] turns recommendation ratings into a daily revision. At the end of each simulated day, it trains a small added set of weights (LoRA) by direct preference optimization, which favors preferred responses over less preferred ones. The revised model then serves later recommendations. The ratings come from simulated users, and no check is reported before the revision takes effect.

Revising prompts and written instructions. Text that the model reads can be revised without training, and we distinguish the reported methods by how they decide which text to keep. PACE [599] takes candidate prompts from a separate proposer and decides only which to commit. A candidate replaces the current prompt when its paired score against that prompt crosses an anytime-valid threshold, which bounds the chance of keeping a change by luck even when the scores are inspected repeatedly. It is tested on three reasoning benchmarks with one model family, and its bound holds per candidate, not per run. In contrast, EmbodiSkill [600] first separates faulty instructions from failures to follow valid ones and then revises its skill files, the written procedures that Action follows. It rewrites defective instructions but only notes execution lapses in an appendix to the skill, so a failed action need not change a valid rule. Its revisions do not have to

pass a check of outcomes before they take effect. In a running assistant, EAIA [411] rewrites its drafting prompts from the user's edits, and later drafts use them (§7.2). Unlike EAIA, Manus proposes updates to a project's instructions and files from patterns in completed tasks and applies them only with the user's authorization [408].[1]

Revising storage and retrieval rules. Feedback about memory can call for an update to the running state or for a revision of the mechanism. In Memento [231], a user who dismisses a recurring recall stops that item from recurring, which corrects retained content. EvolveMem [598] instead revises its retrieval rules and reverts any revision that lowers its score in offline tests on two memory benchmarks.

Revising tools and APIs. In a personal assistant, a revised tool usually changes outside effects. Such a revision therefore needs one more piece of evidence: that the messages it sends or the entries it writes stay within what the user currently authorizes. Among the components of the implementation, however, tools are revised least often in the works we reviewed. SkillSmith [597] provides an example by revising skills together with the tools they call.

Revising the harness while the assistant runs. The harness runs every step, so rewriting it can also change work that is already under way. MOSS [98], Ouroboros [99], and Enoch [560] rewrite their own code, including their harness. Each rewrite must pass a check before it runs. MOSS repairs its code from batches of failed service transcripts and replays the failed batch. It replaces the running container, an isolated environment holding the software it runs, only after review and the user's consent, and it restores the previous version if a health check fails. Its evidence is limited, however, since repairs are made and graded on the same four tasks and consent is automatic. Ouroboros proposes changes to its core code and prompts from operating experience, and each change passes preliminary checks, a blocking review, and a final recheck. Unlike MOSS, it keeps the live history separate from a fixed benchmark history, and later runs use only reviewed versions. Enoch adds a human step: reviewed changes run only after its custodian merges them.

Which revisions are governed. The two forms of control in §6.3 apply unevenly across these changes. A change of policy within a fixed form is the easiest to bound in advance, as the parameter bounds of the three mobile-health trials show. Changes to text and code are harder, because the control can sit in the same files that revision rewrites. Changes of space need the closest control, since a new skill or tool can act outside the assistant, yet VisualClaw, SkillSmith, and EmbodiSkill report no control that applies the user's limits before such a change takes effect. Because an always-on assistant revises between requests, often while the user is away, a control that revision can reach may be rewritten before anyone notices.

Across components, the reported systems meet the four difficulties unevenly. First, none revises when or what Sensing observes, so evidence that was never observed stays missing: ProPerSim, for example, learns whether to recommend at each step but still decides every 2.5 minutes. Second, we found none that reports how a revision changes what the assistant records afterwards. Third, most revisions rest on feedback that names its decision, such as ratings, edits, and dismissals, and only the mobile-health trials learn from delayed outcomes, which they can attribute because they randomize each decision. Fourth, none shows the last step, later use, in full: what happens to commitments made under the old version, or whether later assistance improves by measures independent of the training signal. Indeed, an accepted revision need not bring a benefit [601].

[1] Manus Projects: https://manus.im/en/blog/manus-projects-self-updating

7 HALO: Harness for Always-On Loop Oversight

In this section, we introduce HALO, a diagnostic tool that assesses the loop completeness of existing systems.[1] It examines two parts of the framework of §2: the coupling channels of the continuing loop and, for self-evolution that rests on outcome feedback, the five steps of the chain of evidence. For each channel or step, it asks two questions. Does a system's source report it? And does removing it change the system's behavior? The five steps, defined in §2.1, are record, outcome, attribution, revision, and later use. For an email assistant that learns from the user's edits, as EAIA [411] does in §7.2, the record is the draft it wrote and the outcome is the user's edit. Attribution links the edit to that draft, the revision rewrites the drafting prompt, and later use is the next draft written with the new prompt. HALO answers the two questions in two modes. (1) In *read mode*, language-model reviewers read a system's papers and released code and decide whether each channel and evidence step is reported. (2) In *run mode*, we run a system's released code on fixed inputs, remove one or more channels or evidence steps, and examine how its behavior changes. The word harness in its name refers to these two modes of analysis, not to the harness inside an assistant's scaffolding. We first present and analyze the read-mode results for 40 works, and then demonstrate run mode on two executable cases.

7.1 Read Mode: Channel Coverage and Evidence for Self-Evolution

We apply HALO in read mode to 40 works, listed in Table 22. For each work, read mode gives the same paper or blog to three oversight agents, language-model agents that act as reviewers: GPT-6-Astra, Claude Opus 5.5, and Claude Fable 5.1, all at maximum effort. The agents read the code alongside the text when code is available, falling back to text alone otherwise. Each oversight agent gives its own judgment for every channel and evidence step, and a majority vote of the three agents decides each cell of the table.

The table follows the two parts of our framework. The coupling-channel columns show which information flows of the continuing loop each work reports, while the chain-of-evidence columns show the completeness of evidence for the self-evolving loop. In read mode, we focus on the five additional channels, omitting the basic S→A path common to every agent system. In these columns, ● denotes a reported implementation, ○ an explicitly absent flow, and ◑ insufficient reporting to decide. The chain-of-evidence columns follow the five steps defined in §2.1: record, outcome, attribution, revision, and later use.

Memory guidance is more often reported for Action than Sensing. Across the 40 works, S→M and M→A each have 37 reported implementations, A→S has 28, and M→S has 17. The first two counts reflect a common pattern among these works: the assistant retains observations and retrieves them when deciding how to help. Through M→S, retained state also guides which observations Sensing acquires or how it interprets new input. A system can therefore use personal history extensively when choosing an action without using that history to guide new observations. For a continuing loop, this distinction determines whether accumulated experience also guides the search for fresh context. The lower M→S count identifies a gap in reported coupling: of the remaining 23 works, 22 provide insufficient detail and only 1 explicitly lacks the flow. Missing documentation therefore leaves one question open: whether these systems use retained state to decide what to observe between requests. The counts therefore measure what sources report, not what systems do: a missing mark can mean a missing flow or a missing description. Our analysis in §3.4.2, however, explains why this channel is hard to build and to test, so the gap may not be only a matter of documentation. Because an always-on assistant must notice relevant changes without waiting for a request, we see M→S as a priority for future research.

Action records provide only part of the feedback chain. Most reviewed works keep records of their actions, but few go on to revise their mechanism from them. Of the 40 works, 25 report the A→M channel, which passes action context to Memory, and only 15 of them also retain the outcomes. The other 10 mostly pass plans, responses, or execution records, with no evidence that later consequences are kept. In the chain-of-evidence columns, which count all 40 works, 24 keep action records, 16 retain outcomes, and 18 link outcomes to decisions, whereas only 5 revise their mechanism and only 4 show the revised mechanism in

[1] The full HALO codebase is available at https://github.com/Always-On-Intelligence/HALO.

Table 22 Channel review and chain of evidence for self-evolution. Left: coupling channels, ● reported, ○ explicitly absent, ◑ unreported or insufficiently specified. Right: the chain of evidence of §2.1, ■ stated in the primary source, · not stated. **Rec.** record, **Out.** outcome, **Attr.** attribution, **Rev.** revision, and **Use.** later use.

System	Coupling channels					Chain of evidence					Source
	S→M	M→A	A→S	M→S	A→M	Rec.	Out.	Attr.	Rev.	Use.	
Wearable											
Vinci [8]	●	●	◑	◑	◑	·	·	·	·	·	Report Code
Vinci2 [283]	●	●	◑	◑	◑	·	·	·	·	·	Report Code
VisionClaw [10]	●	●	●	◑	●	■	■	·	·	·	Report Code
ContextAgent [96]	◑	◑	●	◑	◑	·	·	·	·	·	Report Code
ProAgent [130]	●	●	●	●	●	■	■	■	·	·	Report –
AiGet [247]	●	●	◑	●	●	■	·	·	·	·	Report Code
Alpha-Service [131]	●	●	●	◑	●	■	·	·	·	·	Report –
Memento [231]	●	●	●	●	●	■	·	■	·	·	Report –
SocialMind [246]	●	●	◑	●	●	■	·	·	·	·	Report –
SpeechLess [134]	●	●	●	●	●	■	·	■	·	·	Report –
M3-Agent [95]	●	●	◑	●	◑	·	·	·	·	·	Report Code
LightMem-Ego [137]	●	●	◑	◑	◑	·	·	·	·	·	Report Code
Sensible Agent [418]	○	○	●	○	○	·	·	·	·	·	Report –
Egocentric Co-Pilot [151]	●	●	●	◑	◑	·	·	·	·	·	Report Code
Pro^{2}Assist [145]	◑	◑	◑	◑	◑	·	·	·	·	·	Report –
Pro^{2}Bench [237]	●	●	●	●	●	■	■	■	·	·	Report –
IPIBench [129]	●	●	●	●	◑	·	·	·	·	·	Report Code
Omi† [138]	●	●	●	●	●	■	·	·	·	·	Report Code
Bee† [310]	●	●	◑	◑	●	■	■	·	·	·	Report –
Physical											
AssistantX [2]	●	●	●	●	●	·	·	■	·	·	Report Code
Hobbit [3]	●	●	●	●	◑	·	·	■	·	·	Report –
STRANDS [4]	●	●	●	●	●	■	■	■	·	·	Report Code
PHILIA [5]	●	●	●	◑	●	■	·	■	·	·	Report –
ABot-Claw [6]	●	●	●	◑	●	■	■	■	·	·	Report Code
MoveCare [158]	●	●	●	●	●	■	■	■	·	·	Report –
ElliQ† [319, 362]	●	●	●	◑	◑	·	·	·	·	·	Report –
Digital											
PASK [468]	●	●	◑	◑	◑	·	·	·	·	·	Report Code
User as Code [302]	●	●	◑	◑	◑	·	·	·	■	■	Report Code
PITCH [248]	●	●	●	◑	●	■	■	■	·	·	Report –
Ouroboros [99]	●	●	●	◑	●	■	■	■	■	■	Report Code
VisualClaw [185]	●	●	●	◑	●	■	■	■	■	■	Report Code
Proactive Agent [12]	●	●	●	◑	◑	■	■	·	·	·	Report Code
PAHF [232]	●	●	●	◑	●	·	■	■	·	·	Report Code
AwareLLM [14]	●	●	◑	●	◑	·	·	·	·	·	Report –
ProAct [152]	●	●	●	●	●	■	·	·	·	·	Report Code
Long-term TOA [315]	●	●	●	●	●	■	■	■	·	·	Report –
ProPerSim [369]	●	●	◑	◑	●	■	■	■	■	■	Report Code
Dot† [16]	●	●	●	●	●	■	■	■	·	·	Report –
Muse† [15]	●	●	●	◑	●	■	·	·	·	·	Report –
EAIA† [411]	●	●	●	◑	●	■	■	■	■	·	Report Code

Wearable, Physical, and Digital identify the principal interaction domain, and domains may overlap. The basic S→A path is not shown because it is common to every agent system. † Product documentation or an open-source release.

later use. The steps are counted separately, so a later step can be reported more often than an earlier one. More works link outcomes to decisions than retain them, because an immediate check can connect a result to its action without storing the result. Conversely, a stored outcome does not by itself show which earlier decision produced it. For an always-on assistant, action records can already support resuming interrupted tasks and avoiding repetition. A claim of self-evolution, however, requires evidence that experience leads to a mechanism revision and that the revision shapes later behavior.

ProPerSim [369], placed at L4 in Table 21, is one of the few works with such evidence: it is one of only three works that state all five steps, and it shows how the steps connect. It records each user action together with its own response, which can be a recommendation or a choice to stay silent (record). A simulated user rates candidate responses (outcome), so each rating is tied to the response it evaluates (attribution). At the end of each simulated day, the rated pairs train the model by direct preference optimization, a training method that raises the probability of higher-rated responses relative to lower-rated ones (revision). From the second day on, the trained assistant scores above the untrained baselines while recommending less often (later use). The chain is therefore complete, but only against a simulated evaluator, and its delayed effects on real users remain unknown. The source also reports no control that applies the shared conditions to these revisions, so the evidence does not establish L5. A complete chain also does not by itself make a self-evolving loop, which requires an always-on assistant at L3 or above. VisualClaw [185], for example, states all five steps but acts only when asked, so it is a self-improving agent in the sense of Table 3 and stays at L2.

Channel coverage and self-evolution are distinct. No work that reports every channel in the table also reports mechanism revision. Conversely, the three works with a complete evidence chain, Ouroboros [99], VisualClaw [185], and ProPerSim [369], leave some channels unspecified. The table thus captures two different kinds of progress. Wider channel coverage means a more connected continuing loop: retained state steers what the assistant observes and does, and the results of its actions return to Memory, so assistance can continue between requests without the user restating context. A more complete evidence chain means better grounded self-evolution: the assistant can show that a change to its mechanism came from the outcomes of its own decisions and that its later behavior used that change. The first describes how well the assistant keeps running, the second how reliably it improves from use. Neither dimension alone establishes reliable personal assistance across changing goals and interaction boundaries. User as Code [302] further illustrates why the evidence steps are separate: it revises alert rules and uses them later without a reported chain from earlier action outcomes. Combining channel and evidence marks into a single score would obscure how these dimensions and steps differ.

Channel implementations need tests of their contribution. The same system can receive different channel judgments depending on how it runs. VisualClaw [185], for example, can run on two coding agents, Claude Code and Codex. Only Claude Code checks the workspace during execution, while Codex works from a snapshot taken before the call, so only the Claude Code setup reports A $\rightarrow$ S. A channel judgment therefore depends on which setup delivers the signal, when it arrives, and which interface receives it. The evidence behind the table also varies: some sources compare modules, feedback conditions, or whole systems, while others describe product behavior without a controlled comparison. Such evidence can establish an implementation or a broader system benefit while leaving the contribution of an individual channel unresolved. Removing a channel does not always reveal that contribution: another route can supply similar information, so the output can stay unchanged even when that information matters. Revisions, too, can be implemented without being shown in use: EAIA [411] reads revised prompts in its code, but the source provides no run demonstrating later use, leaving the final step unmarked. Run mode tests these contributions directly, separating what a source reports from what changes when a delivered signal or evidence step is removed.

7.2 Run Mode: Testing Channels and the Chain of Evidence

Run mode works like an A/B test: we first run the unmodified system, then rerun it with selected channels or evidence steps cut while keeping the test inputs and the model fixed. For a channel, comparing the two runs shows whether the other modules still respond normally without the cut information. For an evidence step, the cut is made while the system learns from feedback, and later tasks show whether what it learned still carries over. We apply run mode to AiGet [247] to test channels and to EAIA [411] to test evidence steps,

following their released codebases. Claude Opus 5.5 builds and runs the tests, and GPT-6-Astra judges the outputs without knowing the condition, that is, whether an output comes from the unmodified or the cut run.

AiGet is the L3 smart-glasses assistant in Table 21, and it shows why channels need to be cut both singly and together. In its pipeline, the same user profile enters both the perception prompt, which interprets the scene, and the generation prompt, which writes the suggestions, so the profile reaches Action by two paths. Cutting one path can then leave the output unchanged even when the profile matters. The tests cover 6 everyday scenes rendered as text, with 5 samples per scene and condition, and the pipeline runs on Claude Opus 5.5. In Figure 11a,b, we test (1) $\mathrm{M}\to\mathrm{A}$ by withholding the profile from generation while retaining the original perception output, (2) $\mathrm{M}\to\mathrm{S}$ by withholding it from perception, and (3) $\mathrm{A}\to\mathrm{M}$ by withholding the recording of shown suggestions before revisiting the scene. We also cut $\mathrm{M}\to\mathrm{A}$ and $\mathrm{M}\to\mathrm{S}$ together by withholding the profile from both prompts. Of the remaining channels, we do not test $\mathrm{S}\to\mathrm{M}$ because the profile is preloaded before the run, and AiGet does not implement $\mathrm{A}\to\mathrm{S}$. We keep the basic $\mathrm{S}\to\mathrm{A}$ path intact so that Sensing can still pass the interpreted scene to Action.

Without the record of shown suggestions, AiGet repeated earlier suggestions when it revisited a scene, and unrelated history did not provide the same protection. In the profile tests, removing the profile from perception made some settings appear unfamiliar. Yet removing it from either prompt alone left the judge without a clear preference between the modified and original suggestions. When both prompts lost the profile, the judge preferred the original suggestions. Together, the single and joint cuts suggest that the profile's two paths to Action can compensate for each other. When $\mathrm{M}\to\mathrm{A}$ is cut, profile-informed perception can still reach Action through $\mathrm{S}\to\mathrm{A}$, and when $\mathrm{M}\to\mathrm{S}$ is cut, generation can still use the profile directly. A change in scene interpretation can thus coexist with no clear difference in final preference. When paths compensate in this way, cutting one channel at a time locates effects along a path, while cutting two channels together can reveal a dependence that the other path masks. A channel's contribution can therefore depend on the other paths in the loop. This is one reason to examine integrated systems as whole loops rather than only one module at a time.

For self-evolution, we test whether revisions made while an assistant learns from a user's edits shape its replies to new emails. We reproduced EAIA [411], the L4 email assistant in Table 21. It revises its prompts from edits to its drafts through a process called reflection. In the learning phase, a simulated user with 5 unstated preferences edits drafts for 5 emails, and the model that builds the tests also simulates this user. In the test phase, 4 new emails check whether those preferences carry forward without further guidance. In Figure 11c,d, each intervention cuts the chain during learning. For the first two steps, (1) reflection loses the assistant's questions and drafts (record), and (2) user feedback is withheld from reflection (outcome). For the remaining steps, (3) current feedback is paired with the previous email's record, skipping reflection for the first email (attribution), and (4) revised prompts are generated but not stored (revision and later use, tested together). To see whether the effects hold across models, EAIA itself runs on three models, GPT-6-Astra, GPT-6.1-Sol, and Claude Opus 5.5, with 2 runs per condition. The judge then compares the test-phase replies with those of the unmodified run. A test counts as changing the output when the unmodified run is preferred in at least 5 of 8 comparisons for at least one model. When feedback was withheld or revisions not stored, all 3 models dropped the required closing sign-off from new emails, and partnership replies no longer mentioned the designated colleague. By contrast, the record and attribution interventions had less consistent effects across models. Read mode leaves later use unmarked in Table 22 because the source reports no run demonstrating it. The run-mode tests supply this missing evidence: subsequent replies in the reproduced configuration depend on stored revisions.

We designed HALO as a general interface, so that research teams can diagnose their own assistants with the framework we propose. To do so, we release an MCP plugin and an adapter interface. Through the MCP plugin, a running assistant logs each step of its loop and the earlier steps it used, and HALO reads from these links which channels and evidence steps the trace shows. Through an adapter, a team declares where each channel and evidence step passes through its code, and HALO runs the test cases with them cut and reports how the outputs changed. A channel or step that the adapter does not declare cannot be cut, so HALO reports it as not tested. §8.4 discusses what cutting channels in a running assistant would require.

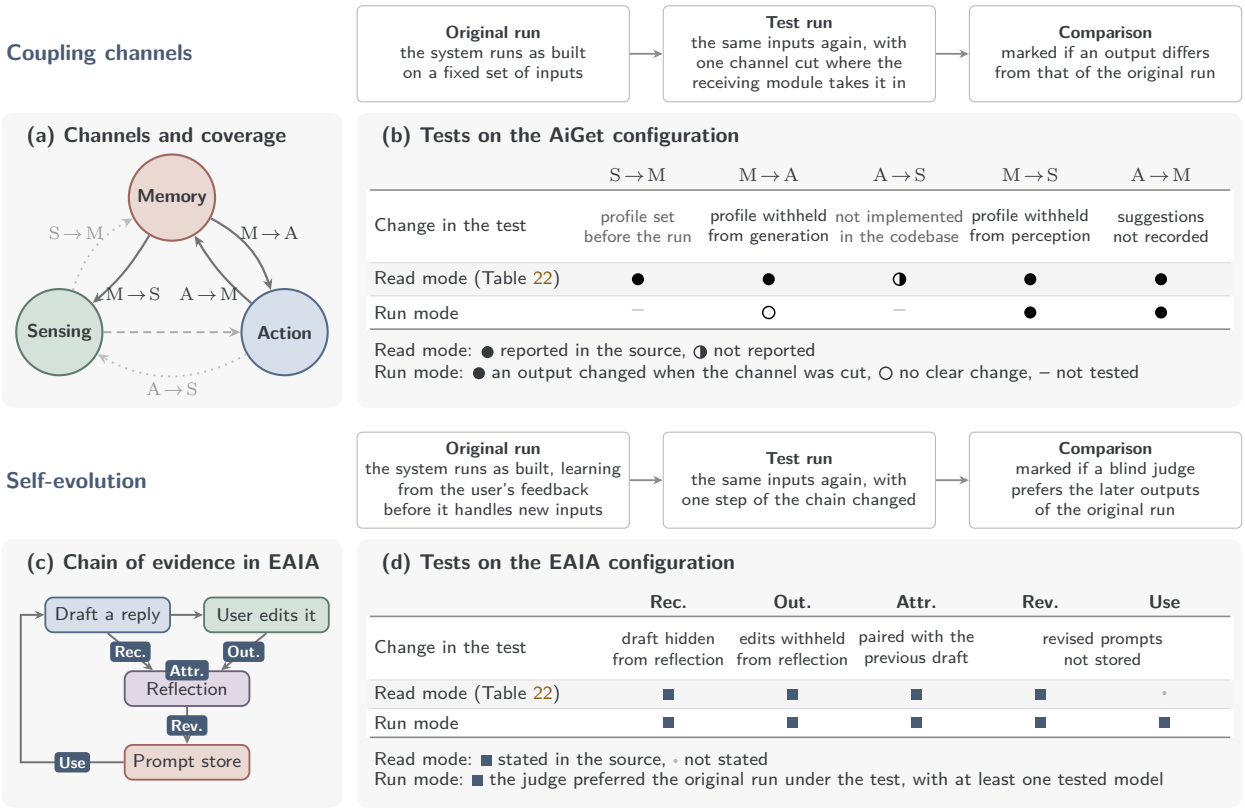


Figure 11 Run mode on AiGet and EAIA. Top: (a) AiGet's coupling channels and (b) the results of the channel tests. Bottom: (c) EAIA's learning loop with the five evidence steps and (d) the results of the evidence-step tests. In (b) and (d), the read-mode row repeats the system's row in Table 22, and the run-mode row marks tests that changed the output.

8 Trends and Open Problems

The preceding sections (§§3–7) show where current always-on systems fall short, and this section states the resulting gaps as open problems. We group the problems into four parts, summarized in Table 23. The first three follow the framework from running the loop to revising it: sustaining the continuing loop, keeping it within authority, privacy, and safety, and revising it through self-evolution. The fourth asks how to evaluate all of this over months of real use. Each problem first states why it remains open, then poses it in a box, and then gives directions and a way to test them.

8.1 Sustaining the Continuing Loop

We first examine how the assistant keeps the loop running when no one is making a request. The problems follow the loop from Sensing through Memory to Action: observing between requests, updating what the assistant believes about the user when no one announces a change, keeping and closing commitments, and deciding when to act, wait, or ask. Because all three modules draw on one budget, the last problem asks how to share it across pending tasks.

8.1.1 Observing Between Requests

Memory guidance of observation is less reported. A request-only assistant observes when the user asks, and the request shows where to look. An always-on assistant has to decide for itself when to observe and what to look at, since real-world limits rarely let it observe everything. Retained state can guide where and when it looks, but it brings a risk that a request does not. When the user's situation changes after it was

Table 23 Open problems at a glance. Each part is one subsection and states its problems in numbered boxes. The *Framework* column names the part of the framework in §2 that each part concerns.

Open problems	Framework	Section
Sustaining the continuing loop. How can an assistant decide what to observe, what to believe, and when to act when no one is asking, while pending commitments share one budget? (Problems 1–5)	π_S, π_M, π_A, the six channels, and c_t	§8.1
Authority, privacy, and safety. What has the user authorized, where does the assistant run, whose information does it hold, and how can people check what it does when no one is watching? (Problems 6–9)	c_t	§8.2
Self-evolution and recursive revision. How can an assistant learn from implicit, delayed outcomes, show that a revision helped, and keep revisions, including revisions of the revision procedure, within the user's bounds? (Problems 10–13)	θ_t, REVISE, A→M	§8.3
Longitudinal evaluation and real deployment. How can we evaluate an assistant whose decisions shape its later inputs, and show over months that it helps the user more than it burdens them? (Problems 14–15)	The whole loop over months	§8.4

recorded, the assistant can keep trusting the old record and stop looking where the change happened. A missed observation then leaves nothing to check. In the running example, a meeting moved by email leaves the stored end time looking correct until the assistant looks again. The risk is larger in an open world, where much of what matters was never recorded, such as a new person, place, or task. Observation that follows memory can then stop exploring what the assistant has not yet seen, even when every record is still correct.

Such guidance is reported less often than memory guidance for Action, and the reviewed tests do not count what it causes the assistant to miss. Only 17 of the 40 works that HALO reviews report M→S, whereas 37 report M→A. Tests that let beliefs go out of date score what the agent asks and judges [232, 233], but a skipped observation leaves nothing to score.

Open Problem 1

A request-driven agent looks only where the user points, whereas an always-on assistant relies on its own memory to decide when and where to look between requests. Yet if a stored belief goes stale, it can steer sensors away from the very place where a silent change just happened. **What keeps memory-guided sensing from blinding the assistant to unannounced changes, and how can the assistant account for the stretches of time it chose not to observe?**

Directions. Memory guidance makes observation cheaper only while the memory is right. A first direction is therefore to save effort by checking only when the assistant is uncertain [197], but this creates a dilemma. Because an unannounced change in the world triggers no warning, the assistant stays wrongly confident about a belief that has actually changed and never learns that it needs to look [232]. It notices the mistake only when something goes wrong or the user complains, which is usually too late. Checking everything on a regular schedule can catch such changes, but it wastes resources and, when a check means asking, keeps interrupting the user. The assistant thus faces a trade-off: it either misses silent changes because it trusts its memory too much, or it interrupts the user by repeatedly checking beliefs that are still right. A second direction is to make occasional checks that do not depend on memory guidance, which can reveal a change that no retained state points to or something new that memory has never contained. Both kinds of check cost energy and can expose private information, so they compete with other pending tasks for the same budget. Comparing the two directions requires an independent record of what happened, so that missed events can be counted under a matched observation budget. Such a record is costly to collect over long use and can itself capture private information. The comparison would also follow each miss into the later decisions it affects.

8.1.2 Updating Beliefs Without Being Told

Changes often arrive without an announcement. Assistants with durable memory already update stored facts when the user states a change. Mem0 [295], for example, decides for each fact extracted from a conversation whether to add it, update a stored memory, or delete one it contradicts. LoCoMo [374] and

LongMemEval [375] make such updates comparable across systems. An always-on assistant often learns of a change only from what it observes, and it can reuse a mistaken belief between requests before the user notices. It must then decide what each observation means for a stored belief.

The evidence for an unannounced change is often weak. Sometimes a change shows up only as something that stops happening, which leaves nothing new to observe. When a deviation does appear, it rarely says how long the change will last: a profile item can be standing, temporary, or expired, and one deviation can fit more than one of these. For example, if the user skips the usual morning briefing for two weeks while working from another city, the assistant has to judge whether the routine has changed or the trip is an exception. Overwriting the old belief too early discards an alternative that has not been ruled out, while waiting keeps an outdated belief in use. Even after the assistant judges correctly, an update repairs only the records it reaches, since summaries, later responses, and cached reads can still depend on the old value [357].

Open Problem 2

People rarely announce that their habits have changed, so the assistant simply observes a deviation, such as a morning briefing skipped for two weeks during a business trip. **When a user silently breaks a routine between requests, how does an always-on assistant tell a lasting change from a temporary exception—and once it updates that belief, what ensures the correction reaches every summary and pending task built on the old value?**

Directions. The assistant has to judge when a deviation should change a stored belief, and the change must then reach everything built on the old value. A first direction is to act on new evidence once it arrives. The method in STALE [233] marks, replaces, or blocks outdated records, but it waits for relevant evidence. A second direction is to keep the original episodes, so that a deviation can be reinterpreted once its duration becomes clear. Such broad writing, however, carries uncertain interpretations into later decisions, and routine interaction can poison durable state that later assistance relies on [339]. Selective writing reduces storage and exposure but can discard the evidence needed to resolve a later contradiction. A third direction is to link each summary and cached read to the records it was derived from, so that a correction reaches them before they are reused. Tests of these directions would deliver changes only through observation, some lasting and some temporary, and score how long an assistant acts on an outdated belief, how often it overwrites a belief that still holds, and whether a correction reaches derived records in time. Memora [337] tests a related question: whether invalidated memories still affect later behavior.

8.1.3 Keeping and Closing Commitments

Commitments outlive the conversation. A request-only assistant with durable memory can hold a reminder at a time the user fixed, which carries out the earlier request as asked. A long-horizon agent carries dependent steps toward given tasks, possibly across sessions or a stream of tasks. An always-on assistant also holds commitments whose timing the user has left to it, and such commitments can outlive the interaction, the device session, or the permission under which they were made. The report reminder is an example: it waits for the end of the meeting, which can move or be cancelled, and the assistant can be asleep or restarting when the meeting ends. Each change has to reach the reminder before it fires.

Resuming a commitment must neither repeat nor skip work. LIMBO [243] tests duplicate effects within a task, not across interactions. In its tests, leading models rarely duplicate a write when immediately reading the resulting state reveals what happened. However, they duplicate it in 56% and 74% of episodes when the request is still in flight, that is, sent but not yet answered, or was delivered twice.

Closing a commitment raises a further problem. The hardest case is a completion that leaves no trace the assistant can observe, and retiring a commitment on incomplete or ambiguous evidence of completion remains largely unevaluated.

Open Problem 3

Unlike a single-session task, an always-on commitment can sit waiting for days while the target event moves, the assistant restarts, or the user quietly finishes the job on their own. **Once a commitment outlives the conversation that created it, what prevents a restart or schedule change from dropping or duplicating the action, and how does the assistant close a commitment whose completion leaves no observable trace?**

Directions. A restart, a moved event, and work the user finishes without saying so each call for a different fix. A first direction is to keep a record of each past action with its commitment, so that a resumed task can see what was already done. In HALO's run mode, AiGet [247] repeated earlier suggestions when it revisited a scene once this record was withheld. The record must also outlast the program that runs the loop. In the design of DeepSeek's Super-agent [544], each session's state lives in a database, apart from that program, so the program can be restarted or replaced without losing the task. DeepSeek Harness also offers tools that report at once that an action has started and deliver its result later as a new event, so an action still in flight stays visible. A second direction is to route each change to the commitment it concerns, so that a moved meeting reaches the reminder waiting for it. In DeepSeek's Super-agent, each task's session follows the chat topics and pull requests the task depends on, and their later events go straight to it. A change that arrives elsewhere, such as a separate email that moves the meeting, must still be matched to the right commitment. DeepSeek leaves that match to the session that talks with the user. A third direction is to close a commitment only on evidence of completion. Pally [312] closes a bill reminder when a verified payment confirmation arrives, and Folk [358] asks the user for a screenshot or a one-line confirmation before closing a task. A test of these directions would disturb a commitment while it is pending: restarting the assistant, delivering a request twice, moving or cancelling the awaited event, and completing the work without telling the assistant. It would count repeated, skipped, and stale actions and measure how long a fulfilled or cancelled commitment keeps producing reminders.

8.1.4 Deciding When to Act, Wait, or Ask

Outcomes arrive after the decision. An assistant that answers requests acts when asked, and the user is present to respond. An always-on assistant decides for itself whether to act, wait, or ask, and the outcome of that choice often arrives much later. Waiting can avoid an unnecessary interruption, but waiting too long can miss the opportunity or leave the commitment unfulfilled. Asking has a cost as well, since a question can interrupt ongoing activity [121, 442].

The value of acting also changes with repetition. In the six-week HeartSteps trial [433], walking suggestions at first raised steps in the following 30 minutes by 107%, but by 24% on average across the whole trial. The trial does not identify fatigue as the cause. Timing benchmarks assess when an intervention occurs, yet useful assistance today can become burdensome after repeated use.

Open Problem 4

Every unprompted message spends the user's attention immediately, whereas the cost of staying silent may surface only days later as a missed deadline. Moreover, suggestions that help in week one can turn into annoying noise by week six. **In unprompted interactions, how does an always-on assistant weigh immediate interruption cost against delayed benefit as user fatigue builds up over weeks?**

Directions. A good choice of when to act depends both on what has already happened and on what will follow. A first direction is to choose among silence, communication, preparation, and authorized execution, as abstention and proactive-assistance studies propose [40, 41], while conditioning the choice on the interruptions already accumulated and on the user's receptivity, meaning readiness to receive help. A second direction is to learn from experts who already make such choices, not only from the user's reactions. Learning from the user alone spends the user's attention on every attempt and waits for outcomes that may arrive days later, whereas expert records already show when skilled people step in. In training, this takes the form of

imitation learning: a model is trained on records of when these people spoke and what they said. Live fitness coaching [602] trains a model on feedback written from a coach's perspective, and live commentary [223] trains one on the narration of online videos. The instructor interventions recorded in HoloAssist [603] show when and how a remote expert steps in.

At the time of use, the assistant can learn from experts through in-context learning. Before guiding a user through a task, for example, it could watch an expert's online video of that task and learn when to step in and what to say. Demo-ICL [604] tests this with how-to videos, where a model watches a demonstration and predicts the next step in another video of a similar task. The strongest model it tests answers 36% of these questions correctly, against 80% for people. Robots can also learn from demonstrations in context [605], and RoboICL [606] has a general model output the robot's motions directly. Its demonstrations, however, are still recorded on the same robot with a person steering it, as in ICRT [607]. Bridging the gap between online video and action in the physical world therefore remains far off.

A third direction is to train the decision on later outcomes as well as on immediate acceptance. Reward-design work such as ProRL [608] shapes objectives that encourage activity, yet frequent acceptance reflects only the immediate response, so such objectives still need tests of whether they improve user benefit. Evaluating these directions needs studies that repeat the decision over weeks and record what follows each choice, including the choice to wait, and that weigh user benefit against interruption costs, missed opportunities, and unfulfilled commitments. The repeated randomization in HeartSteps shows that such studies can be run in the field, and Problem 10 asks how their outcomes can revise the decision rule.

8.1.5 Sharing Limited Resources Across Pending Tasks

Pending tasks draw on one budget. Every module draws on the same battery, computation, and model calls, and pending commitments compete for this budget. On smart glasses, for example, one battery powers both the camera that observes and the display that shows the reminder. A device that serves one request at a time spends the budget on that request. An always-on assistant, in contrast, holds several pending tasks with different observation needs. During a familiar commute, the assistant can sample the camera less often to save energy. An unresolved action instead needs a fresh check of its outcome, and that check can delay checks on other pending tasks.

Two inputs to Sensing can then favor different observations: memory guidance through M$\rightarrow$S and an action's request for a check through A$\rightarrow$S. With overlapping tasks, focusing observation on one task can also reduce chances to detect new events or follow other commitments.

Compensation adds a less visible risk. When selective sensing omits evidence, a task log or broader monitoring can recover it, but both methods can depend on the same resource or permission and fail together. To confirm that a message was sent, for example, the assistant can read the outbox and fall back on its task log, yet both rely on the same cloud account. A shared dependence can also stay hidden in tests. In HALO's run mode, both of AiGet's paths to Action read one user profile, and the results suggest that the two paths can compensate for each other. Cutting either path alone left HALO's judge without a clear preference, whereas cutting both made it prefer the original suggestions.

Open Problem 5

A single-task agent spends the whole device budget on the active request, but an always-on assistant juggles multiple pending commitments that all compete for the same battery, computation, and sensors. **When multiple background commitments compete for one observation budget, which checks take priority—and if a backup check, such as reading a task log, covers for a skipped sensor check, how can the assistant detect that both rely on the same revocable account or permission?**

Directions. To share the budget, the assistant has to know which check can wait and which method can cover for it. A first direction is therefore to set the priority between the two inputs to Sensing. One design we propose gives priority to an action's fresh checks of unresolved effects and lets one observation serve both

inputs when their viewpoint, modality, and freshness needs match. Observation must then change again when a task ends, changes, or is cancelled. A second direction is to record which resource or permission each method depends on, so that a method chosen to cover a skipped check does not fail for the same reason. An evaluation would run overlapping tasks with newly arriving events under a fixed observation budget, vary agreement and conflict between memory guidance and action requests, and vary whether covering methods share a resource or permission. It would measure evidence freshness, task completion, and missed chances to correct a belief. Cutting channels jointly as well as one at a time would show both what a covering method recovers and what it hides.

8.2 Authority, Privacy, and Safety

We next examine three of the shared conditions: authority, privacy, and safety. We single them out because they limit what the assistant is allowed to do, not what it is able to do. People set the first two, the user above all, along with those the assistant observes or affects, and safety rules out harms that no one's agreement can make acceptable. Each condition must hold in Sensing, Memory, and Action at once, yet no reported control covers all three (§6.2.1). Once the assistant keeps running between requests, these conditions also raise questions about the people involved. What has the user authorized, and for how long? Where does the assistant run, and what can it reach? Whose information does it hold, and how can people check what it does when no one is watching? The CAPS report [609] poses general versions of several of these questions for AI agents, and the four problems below state the form they take for an always-on assistant.

8.2.1 Consent That Outlasts the Conversation

Permission is granted once and used later. In a request-driven assistant, each action that uses a permission follows a request, so the user is present to see it. An always-on assistant instead uses permission granted earlier for actions it chooses later, often while the user is away. Products already differ in where they ask the user again, from Claude Duet's approval before it sends, pays, signs, or deletes to the checks of each action against the user's permissions in Dot and Muse [17, 16, 15].

The CAPS report [609] asks how credentials can be issued for the correct lifetime and revoked when the task completes. An always-on assistant, however, can hold many commitments under one grant, so no single task end marks when the grant should lapse. Changes in permission create a further coordination problem [609], because updating one component can leave another using the old permission. Asking the user, meanwhile, has a cost (Problem 4). An action held for approval can miss its moment, while one that proceeds cannot always be undone.

Open Problem 6

In a turn-based chat, the user is present whenever a permission is used, whereas in a continuing loop, a permission granted once keeps authorizing actions the assistant initiates days later while the user is away. **When a user narrows or revokes a standing permission, what ensures the change immediately halts every scheduled check, queued action, and delegated tool, and which background actions must still pause and wait for fresh human sign-off?**

Directions. A grant checked once when a task is planned says little about an action taken days later. A first direction is therefore to check a grant when the action takes effect. Commit-time authorization [465] does so for general agents (§5.3.4). A second direction, which we propose, is to tie each grant to the commitments that rely on it, so that the grant can lapse when the last of them closes. Every change also has to reach the components that still hold the old permission: scheduled observations, retained instructions, queued actions, and delegated tools. A third direction is to set the level of approval by risk, as the products above do in different ways, so that reversible steps proceed under the user's standing rules while irreversible ones wait. Policies that the user sets in advance, as the CAPS report [609] proposes, could replace approval of each action, although how users maintain such policies over time remains open. Tests would change permissions during overlapping tasks and measure how long affected operations continue, whether the remaining work stays within the new scope, and how much authorized work is interrupted unnecessarily.

8.2.2 Where the Assistant Runs and What It Can Reach

An assistant that acts on its own needs a place to run. Runs that no request starts have to execute after the user closes the app, either on a cloud computer kept apart from the user's devices, as in Gemini Spark, Claude Duet, Dot, and Muse, or on the user's own host, as in OpenClaw [412, 17, 16, 15, 332]. The choice trades speed against reach. An assistant that runs apart from the device it operates sends every command and screenshot across the network, while one that runs on the device acts faster but needs a harness light enough for that device.

A sandbox, however, does not stop the assistant from misusing an interface it has been granted, and isolation can be partial, as when all of a user's Grok Bots share one cloud computer [242]. Other parties set limits too. When ByteDance's Doubao phone assistant began operating other apps on users' behalf, major apps restricted or blocked it, and ByteDance disabled its features in financial and gaming apps [610]. Once the assistant has crossed a limit, stopping it does not undo what it did, since stopping or deleting a Dot leaves its completed actions in place [16].

Open Problem 7

Running an always-on assistant in the cloud keeps it alive when the user's devices are off, while running it locally keeps data on-device and lowers latency. Either way, a code sandbox cannot stop an assistant from misusing a legitimate account it was granted, nor does killing the process undo emails already sent. **Where should each part of the continuing loop run, and what catches or reverses an unauthorized external action when code sandboxes and stop buttons cannot?**

Directions. The boundary has to hold even when the assistant misjudges its own scope. A first direction is therefore to keep credentials and permission checks outside what the assistant can change, as Muse does with separate credential services and with its Sentinel checks [15]. A second direction is to give each task or agent its own environment and identity. Manus gives each task its own isolated cloud virtual machine [408], and Cue and Fo give the agent accounts of its own, so outside services do not record its actions as the user's [611, 459]. Limits on each tool complement such environments: AgentBound [543] confines each server to a declared list of permissions, and IsolateGPT [542] runs each app in isolation. A third direction is to plan for undoing before acting, by recording how each action can be reversed and holding the actions that cannot. Tests would give the assistant instructions that exceed its scope, both from the user and through content it observes, and measure whether the boundary stops them, how long a crossing goes unnoticed, and how much of its effect can be undone.

8.2.3 Whose Context the Assistant Holds

The assistant can help only as far as it sees the user's context. How much an always-on assistant can do depends on how much of the user's context it can see, and on how much the user is willing and able to give it. Running the assistant on the provider's cloud computers lets it work around the clock, but the user's context then also sits on machines that the provider runs. Whatever the assistant holds, a privacy requirement must hold in every module at once, because what one module may process flows through the channels to modules that must not keep it or pass it on. No reported control achieves this [416, 246, 139].

The same information can also be delivered much later, at a time and to an audience that the assistant chooses. A note that is fine to show on the user's phone may not be fine to read aloud in a shared room, and current agents misjudge the audience even within one task [440]. Information from a deleted source can remain in a summary, index, cache, or derived procedure. Observation also takes in other people, whose expectations about being recorded vary with the setting [170, 171, 172, 173], and a continuing loop extends these concerns over months of observing shared spaces.

Open Problem 8

An always-on assistant can capture bystanders who never consented to recording, and a fact captured privately today can be surfaced days later in front of a different audience. **What prevents an always-on assistant from leaking information captured in one context to a new audience days later, and how can deleting a source record purge every summary, cache, and learned routine derived from it?**

Directions. Days after capture, the assistant may no longer know on what terms the information was given. A first direction is therefore to carry its source with each record. A record that keeps its source, owner, permission, and validity lets the assistant ask, at delivery, whether the current audience, place, and purpose fall within what the source allowed, and lets every module check the same terms. A second direction is to carry a deletion along the links between derived records and their sources, so that summaries, indexes, caches, derived procedures, pending deliveries, and copies on the provider's machines stop using a deleted source. A third direction, for other people, is to limit what is captured or answered about them, as Kirigami [140] and Selective Hearing [174] do for speech, and to direct each delivery only to the people who need it. An assistant can also serve a group as a whole, as STRANDS [4] does for the residents of a care facility. None of these directions settles whose consent governs what is kept about other people over months. Tests would change the audience, place, or purpose between capture and delivery and delete a source while derived records and deliveries are still pending. Deployments with several people would record who was captured, what was kept about them, and which actions reached them, and would ask bystanders and group members about the assistant's behavior, not only the primary user [318].

8.2.4 Overseeing an Assistant That Acts on Its Own

People see only part of what the assistant does. Safety concerns the harm that the assistant's actions can cause, and an always-on assistant acts more often while no one is watching. It can follow content written to mislead it, which arrives through the screens, documents, and messages it observes. External text can redirect an agent [168], tool responses can carry untrusted content [167], and injected web content can make an agent leak the user's personal data [441]. What the assistant keeps can steer decisions long after, and summaries can hide the contamination [386]. A retained procedure can also repeat an unsafe shortcut in every later run [530, 531], and an assistant can disregard the limits it was given, as Poke did in one logged run of Assistant Benchmark [489].

People then need to see what the assistant did and why. Outside services record its actions as the user's own, so a later reader cannot tell the user's actions from the assistant's. The user also wants a report short enough to read, yet the item that a short report leaves out may be the one that matters. Reported systems check only parts of this path, and no single control covers it from observation through retained procedures to the action [15, 332, 532].

Open Problem 9

While the user is away for days, untrusted emails or web pages can plant instructions that enter Memory and steer later actions, yet the user only has time to read a brief summary of what happened. **When an assistant operates unattended for days, how can a brief summary expose every high-risk or externally influenced action without overwhelming the user, backed by audit logs and safety checks that the assistant itself cannot edit?**

Directions. Oversight depends on records that the assistant cannot rewrite and on reports that leave nothing important out. A first direction is to trace each action back to the observations, records, rules, and permission behind it. The fields and links proposed for Problem 8 are a start: once planted content is traced to its source, the links from that source would show what to remove. A second direction is to keep observed content from setting the rules that govern later actions. Conseca [541] writes each task's policy only from trusted context, and Progent [540] lets an agent narrow its own tool privileges but sends any widening to the user. A third direction, which we propose, is to report in layers: a short report for the user that lists

irreversible actions first, backed by a review of the full record that flags what the short report left out. That review needs a checker outside the assistant, and HALO's adapter (§7) could provide its interface to a running assistant. Tests would plant misleading content and unsafe instructions and measure their effect days later, after it has passed through Memory into actions and revisions. They would also check whether the trace lets the assistant undo that effect and whether the short report would have shown the risk to the user.

8.3 Self-Evolution and Recursive Revision

We then turn to self-evolution, in which the assistant also revises its own mechanism, and these revisions must stay within the same conditions. An always-on assistant learns from its own operating experience: the evidence for a revision comes from the assistant's earlier decisions, and the assistant keeps serving the user while revising. Each problem below builds on the one before it: linking delayed outcomes to decisions, telling whether a revision helped, keeping revisions within the user's bounds, and revising the procedure that produces revisions.

8.3.1 Learning From Implicit, Delayed Outcomes

Outcomes rarely name the decision that caused them. Self-improving agents learn from feedback such as environment rewards or model critiques. An always-on assistant learns from daily use, where feedback is sparse, delayed, and open to competing explanations. One dismissed reminder says little: the dismissal can reflect the rule, a meeting that ran late, or a report the user had already sent. The assistant must therefore link the feedback to an earlier decision and then decide whether to correct a user record or to revise a rule. A reminder dismissed while the meeting was still running can call for fixing the recorded end time of the meeting, which is a content update. It can instead call for changing the rule for when to send reminders, which is a mechanism revision.

Few reviewed works carry this link through to a revision. Of the 40 works that HALO reviews, 16 retain outcomes and 18 link outcomes to decisions, often through an immediate check that does not store the result. Attribution can also fall into a memory trap, where faulty memory biases how feedback is read and a revision preserves or amplifies the original mistake. Decisions to stay silent produce even less evidence, since a reminder that was never sent leaves no reply to learn from.

Open Problem 10

In daily use, feedback rarely names the decision it concerns: a dismissed reminder could mean the timing rule is flawed, the stored meeting time was wrong, or the user already finished the task. Staying silent leaves even less trace, because an unsent reminder draws no reply. **When feedback arrives hours later—or never arrives because the assistant stayed silent—how does a self-evolving loop attribute the outcome to the right earlier decision and tell whether to fix a stale memory record or rewrite a decision rule?**

Directions. Attribution needs to know which decision preceded an outcome and what would have followed a different one. A first direction is to keep each decision record together with the outcome that follows it, the step at which many reviewed works stop, so that a later outcome can be matched to the decision it concerns. A second direction is to randomize some decisions, as mobile-health trials such as HeartSteps do. Randomization also yields evidence about silence, since the user's behavior after an unsent reminder can then be compared with that after a sent one. Neither direction by itself tells an outdated record from a flawed rule. Tests would therefore check outcomes against a record kept independently of the assistant and report, for each revision, the outcomes that prompted it.

8.3.2 Telling Whether a Revision Helped

An accepted revision need not help later. Judging a revision is harder in an always-on assistant than in offline tests, because a revision changes what the assistant observes and keeps afterwards. A new rule that checks the calendar less often, for example, also changes what the assistant sees. Later evidence therefore

cannot simply be compared with earlier evidence, and the assistant cannot freely replay the user's activities to try both versions.

Only 5 of the 40 works that HALO reviews revise their mechanism, and only 4 show the revised mechanism in later use. Passing a check before taking effect does not establish that later assistance improved [601]. Such checks can also be weak: self-authored tests are unreliable in heuristic self-improving agents [612], and inspecting noisy scores repeatedly raises the chance of keeping a change by luck.

Revisions can also weaken capabilities that the assistant already had. Don't Blind Your VLA [613] finds that further training a vision-language-action (VLA) model on action tasks can weaken its visual-language grounding, the model's ability to link words to what it sees. Checks after a revision therefore need to cover perception as well as task performance. Continual learning studies models that keep learning new tasks over time, and it calls the loss of an earlier capability forgetting. Such losses appear across workflow, skill, model, and memory evolution in self-improving LLM agents [614].

Open Problem 11

Offline agents can replay the same benchmark before and after an update, but revising a live assistant changes what it observes and records in the user's subsequent days. **Because a live revision alters the future observations used to judge it, what evidence proves that a new mechanism genuinely improved assistance without degrading existing perception skills or breaking commitments made under the old rule?**

Directions. The check that accepts a revision can itself be wrong. A first direction is therefore to accept a revision only through tests that bound the chance of keeping a change by luck, as anytime-valid acceptance tests do [599, 615]. Some of this testing can use logged interactions before the user sees a change, as batch methods for mobile health do [64], but only where the logged decisions varied, as when HeartSteps randomized whether to send each prompt [433]. Otherwise, the log cannot show how the user would respond to actions the assistant never took. A second direction is to check capabilities beyond the one being revised, including perception, since the signal that trains a revision rarely measures what the revision weakens. A third direction is to version each revision of the mechanism, so that commitments made under the earlier version can still be honored and a harmful revision can be rolled back. Evaluating these directions requires later interactions with independently checked outcomes and tasks beyond those used to select the revision. PAST-Bench [24], PATH-Bench [390], SEA-Eval [616], and LifeSide [617] provide task streams for such evaluation, and user-conditioned evaluation [618] tests personal agents under interventions over time. In each case, the assistant is judged by its declared user benefit, cost, or tradeoff.

8.3.3 Keeping Self-Evolution Within the User's Bounds

Few reported controls keep revisions within the user's limits. Governed revision applies the shared conditions to every revision before it takes effect, so that no revision can change the limits these conditions set. Of the systems placed in §6.3, only five reach L5, the level of governed revision. All of them are research systems, and each reaches it only under a stated condition. Assistants at L4 link outcomes to decisions but lack such a control, and they show how a control fails when it lies within reach of revision. OpenClaw [332], for example, records lessons in the file that lists which actions need the user's approval, so a revision can rewrite that list.

Revisions can also drift without anyone noticing, as when a skill library changes without a visible failure [619]. A further question is whether revisions preserve the scope of earlier permissions. A rule learned in one setting can persist into another where it is inappropriate [609]. For example, permission to share one work calendar should not become a general rule to share all calendars.

Open Problem 12

Left unchecked, a self-evolving assistant can generalize a narrow permission, such as sharing one work calendar, into a general rule that shares private calendars too, or even rewrite the file that lists which actions require approval. **What stops a self-evolving loop from drifting beyond the permissions under which a rule was learned, and where is the line between routine revisions that can run within preset bounds and scope-widening revisions that require human approval?**

Directions. A control that revision can rewrite does not bound revision. A first direction is therefore to keep the limits outside what revision can change, as the five L5 works do in different ways: Ouroboros [99] protects its governance files from ordinary writes, Enoch [560] leaves every merge to a human custodian, and HeartSteps [559], Oralytics [424], and MiWaves [425] revise only parameters within fixed bounds. A second direction is to record the permission under which each rule was learned and check it before the rule applies in a new setting, so that a rule cannot spread beyond the permission it came from. A third direction is to match the form of control to the change: approval of each change for revisions that widen what the assistant may do, and limits set in advance for routine updates within an authorized scope. Tests would change the purpose, audience, or permission after a revision and check whether the revised assistant still follows the applicable limits. They would also attempt to revise the control itself, as OpenClaw's lessons can, and confirm that the limits stay out of reach. If a revision changes its own checks, those checks need independent verification.

8.3.4 Revising the Revision Procedure

Recursive revision changes the revision procedure. The questions above arise even when the procedure that produces revisions stays fixed. Recursive revision adds a further risk: the procedure that checks revisions can itself change. One way to implement REVISE in Eq. (3) uses a fixed proposer $\mathcal{U}_{\psi_t}$, which proposes revisions, and a fixed verifier V_{ϕ_t}, which checks them. Both belong to the mechanism θ_t. Recursive revision additionally changes one of these two procedures, retains the change, and uses it in a later improvement round [75]. It therefore requires at least:

$$\text{recursive at } t \implies \big(\psi_{t+1} \neq \psi_t\big) \vee \big(\phi_{t+1} \neq \phi_t\big). \tag{4}$$

A claim of recursion therefore rests on evidence that follows a revised procedure into its next use. Recent works differ in whether they change the proposer, the verifier, or neither. Recursive Harness Self-Improvement [620] conditions each revision of an agent's harness on the history of earlier revisions, and Meta^n [621] keeps its meta-operation, the procedure that writes each new layer, fixed. In both, the input to the proposer grows while the proposer itself stays the same. Hyperagents [622] changes the proposer: its meta agent can rewrite itself, and evolved meta agents generate later variants. Its transfer tests hold the meta agent fixed in a new task domain, and parent selection, which picks the agent to build on, stays fixed in the main experiments. The Red Queen Gödel Machine [623] changes the verifier: it freezes its evaluator during each search epoch and lets a stronger successor score candidates in later search. Offline meta-learning, by contrast, learns memory designs or adaptation policies before deployment [624, 625]. In our view, the key distinction is between repeatedly revising the mechanism and retaining changes to the procedure that produces later revisions [43, 626, 56]. We leave recursive revision in always-on assistants to future work for two reasons. First, only five reviewed works reach governed revision, each under a stated condition. Second, a revised verifier can weaken the checks discussed above, so a revision that harms the user or exceeds a permission could pass unnoticed.

Open Problem 13

Recursive self-improvement modifies not only the assistant's mechanism, but also the proposer and verifier that generate and check future revisions. **If a recursive loop can rewrite its own verifier, what anchors the safety, privacy, and authority checks so the assistant cannot weaken the very tests that govern it? Under slow human feedback that cannot be replayed, what criterion tells the search when to stop?**

Directions. When the verifier itself can change, the loop loses the standard against which a change is judged. A first direction is therefore to anchor any change to the verifier in a fixed reference, as the Red Queen Gödel Machine selects each replacement evaluator against a fixed ground-truth anchor and discards scores tied to the old one. In an always-on assistant, the checks that protect the user's authority and privacy would belong to that fixed reference. A second direction is to use the repeated real rounds of an always-on assistant to test a revised procedure, for example by letting two proposer versions propose revisions from the same incoming feedback. Comparisons would cover three settings, (1) a fixed mechanism, (2) revision by a fixed proposer, and (3) revision in which the proposer can also change, and would match initial states, evidence, and total budgets, including search and validation costs. For recursive claims, improved current-task performance differs from improved ability to produce or select later revisions. Evaluation can also examine when search stops because its expected benefit falls below its cost or risk [75].

8.4 Longitudinal Evaluation and Real Deployment

Finally, we examine how to evaluate all of this over months. Each earlier problem ends with what a test of it would measure, and the problems here concern how to run such tests when the assistant shapes its own later inputs and what they should measure for the user.

8.4.1 Evaluating a Loop Whose Decisions Shape Later Inputs

Replay cannot show what the assistant would have caused. Longitudinal evaluation follows unfinished commitments, recovery after pauses, changing preferences, and delayed outcomes. One difficulty is that feedback can change even when the evaluation rule stays fixed. Once a model is deployed, its outputs change the data it later sees, an effect studied as performative prediction [627], and recommender simulations show that past recommendations shape later training data, which they call algorithmic confounding [628]. Personal assistance likewise changes the user's habits, receptivity, and later feedback. Fixing the grading model, the scoring rubric, and the mechanism version of each run therefore makes the comparison procedure repeatable, but it does not fix how users responded to earlier interventions. In this sense, fixed-input replay, which reruns each version on the same recorded inputs, measures sensitivity to recorded evidence. Current evaluations rarely compare versions in closed loop.

Open Problem 14

Replaying recorded logs tests how two versions react to the exact same past inputs, but in real deployment, each version's actions change what the user does next and which events the sensors capture. **Once an assistant's decisions actively reshape its future inputs, how can we fairly compare two evolving versions in closed-loop deployment, and what independent ground-truth records are needed to reveal the events and side effects the assistant itself missed?**

Directions. Evaluating such a loop is costly, and no two runs of it see the same inputs. A first direction is to compare versions in closed loop or in deployment, so that each version's actions shape the responses it receives. A second direction is to keep the comparison procedure fixed and auditable while the inputs change, and DR Tulu [629], RubricBench [630], and studies of changing verification [631, 632] offer relevant designs. A third direction, which we propose, is an evaluation resource that connects concurrent observation demands, task schedules, decisions, and delayed outcomes, and that specifies what the assistant is allowed to capture and retain [265, 345]. Versioned sources, reusable redacted scenarios, and common intervention tools would let researchers compare resource sharing and later mechanism revision. HALO's design can provide the interface for such tools: the same adapter that cuts channels in released codebases can be attached to a running assistant, such as one on smart glasses. There a cut would also change what the user does next, so the comparison becomes closed-loop. Each cut would also withhold information from a real user, so it would need the user's consent and a limit on how much assistance it may cost.

Concretely, the resource we propose would have four parts. The first is multi-week recordings from consenting participants, with time-aligned sensor streams, application events, and every decision the assistant made, including each decision to wait. The second is an event record kept independently of the assistant, such as

participant diaries or annotator labels, so that missed events can be counted. The third links each commitment to its due and closing times and each outcome to the decision it follows, so that resumption and attribution can be scored. The fourth is a consent record that marks which parts may be released, together with redacted scenarios that let other groups replay the same history and attach their own interventions.

8.4.2 Measuring Benefit and Burden Over Months

Benefits fade while burdens accumulate. Effects on the user change over weeks. HeartSteps [433], whose effect faded within six weeks, does not measure long-term health benefit, and reported field studies differ widely in length, for example five days for ProAIDE [13] and 334 days for one older adult with PALRO [320].

Open Problem 15

Short-term benchmarks reward immediate task completion, yet over months of real use, initial helpfulness can fade while interruptions, confirmation fatigue, and subtle privacy or permission leaks quietly accumulate. **After months of daily use, how do we measure whether the assistant's ongoing help still outweighs the trouble it causes? And what checks verify that privacy and permissions have not quietly slipped over time?**

Directions. Benefit and burden change at different speeds, so a measure of either alone can mislead. A first direction is therefore to follow the same users over months and set benefit against burden at each point: whether commitments are completed, how many needed events the assistant missed, how often it repeated a prompt, how much correction the user had to make, and how many confirmations it requested and when.

A second direction is to test how privacy and authority hold over months of use as circumstances change [609]. We propose scenarios with changes of audience, narrowed permissions, deletion requests, and stop commands. Evaluation would track disclosures across interactions, residual use of withdrawn information, background work that continues after a stop, and missed legitimate assistance. These measures would test whether controls remain effective without making useful assistance depend on constant user supervision.

9 Conclusion

In this paper, we argue that the promise of always-on personal intelligence lies in a continuing loop and in a self-evolving loop that extends it. The continuing loop rests on two properties: *state durability*, which keeps personal state across interactions, and *operational continuity*, which lets the assistant decide between requests whether and when to act. The self-evolving loop further revises the assistant's own mechanism from operating experience. We describe the assistant as Sensing, Memory, and Action, connected by six coupling channels and operating under shared conditions such as privacy and user authority. Within this framework, we review each module, trace cross-module bottlenecks in wearable, digital, and physical applications, and place the reviewed systems on cumulative levels of loop closure. HALO then checks which channels and evidence steps each work reports and tests whether removing them changes the behavior of executable systems.

Our analyses lead to three findings. First, Memory guiding Sensing is reported far less often than Memory guiding Action, so it remains open whether accumulated experience guides what an assistant observes between requests. Second, many works pass action context to Memory, but fewer retain the outcomes, and few carry them through to a revision and its later use, which a claim of self-evolution requires. Third, cutting a single channel can leave a system's output unchanged because another path supplies the same information, so a channel's contribution needs tests that cut paths both singly and together. These findings rest on what sources report and on tests of released code, so a missing mark can reflect missing documentation as well as a missing capability. These findings, together with the open problems in sustaining the continuing loop, in authority, privacy, and safety, in self-evolution, and in longitudinal evaluation, set the next steps. Solving these problems would let an assistant gain more room to act as evidence grows that it serves its user well, while it stays within the conditions the user sets. Users would then manage it less and rely on it more, and personal intelligence could fade into everyday life, as Weiser envisioned for the most profound technologies.

References

[1] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–105, 1991.

[2] Nan Sun, Bo Mao, Yongchang Li, Di Guo, and Huaping Liu. AssistantX: An LLM-Powered proactive assistant in collaborative human-populated environments. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3352–3359. IEEE, 2025.

[3] Markus Bajones, David Fischinger, Astrid Weiss, Daniel Wolf, Markus Vincze, Paloma de la Puente, Tobias Körtner, Markus Weninger, Konstantinos Papoutsakis, Damien Michel, Ammar Qammaz, Paschalis Panteleris, Michalis Foukarakis, Ilia Adami, Danai Ioannidi, Asterios Leonidis, Margherita Antona, Antonis Argyros, Peter Mayer, Paul Panek, Håkan Eftring, and Susanne Frennert. Hobbit: Providing fall detection and prevention for the elderly in the real world. *Journal of Robotics*, 2018:1–20, 2018. doi: 10.1155/2018/1754657. URL https://doi.org/10.1155/2018/1754657.

[4] Nick Hawes, Christopher Burbridge, Ferdian Jovan, Lars Kunze, Bruno Lacerda, Lenka Mudrová, Jay Young, Jeremy Wyatt, Denise Hebesberger, Tobias Körtner, Rares Ambrus, Nils Bore, John Folkesson, Patric Jensfelt, Lucas Beyer, Alexander Hermans, Bastian Leibe, Aitor Aldoma, Thomas Fäulhammer, Michael Zillich, Markus Vincze, Eris Chinellato, Muhannad Al-Omari, Paul Duckworth, Yiannis Gatsoulis, David C. Hogg, Anthony G. Cohn, Christian Dondrup, Jaime Pulido Fentanes, Tomas Krajník, João M. Santos, Tom Duckett, and Marc Hanheide. The STRANDS project: Long-term autonomy in everyday environments. *IEEE Robotics & Automation Magazine*, 24(3):146–156, 2017. doi: 10.1109/MRA.2016.2636359. URL https://arxiv.org/abs/1604.04384.

[5] Weiqi Jin, Peijun Tang, Kuncheng Luo, Baifu Huang, Binyan Sun, Haotian Yang, Shangjin Xie, and Jianan Wang. A glimpse into long-term physical coexistence with intelligent robots, 2026. URL https://arxiv.org/abs/2607.11377.

[6] Dongjie Huo, Haoyun Liu, Guoqing Liu, Dekang Qi, Zhiming Sun, Maoguo Gao, Jianxin He, Yandan Yang, Xinyuan Chang, Feng Xiong, Xing Wei, Zhiheng Ma, and Mu Xu. ABot-Claw: A foundation for persistent, cooperative, and self-evolving robotic agents, 2026. URL https://arxiv.org/abs/2604.10096.

[7] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. Towards wearable cognitive assistance. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, 2014. doi: 10.1145/2594368.2594383. URL https://www.pdl.cmu.edu/PDL-FTP/CloudComputing/ha-mobisys2014_abs.shtml.

[8] Yifei Huang, Jilan Xu, Baoqi Pei, Yuping He, Guo Chen, Lijin Yang, Xinyuan Chen, Yaohui Wang, Zheng Nie, Jinyao Liu, Guoshun Fan, Dechen Lin, Fang Fang, Kunpeng Li, Chang Yuan, Yali Wang, Yu Qiao, and Limin Wang. Vinci: A real-time embodied smart assistant based on egocentric vision-language model, 2024. URL https://arxiv.org/abs/2412.21080.

[9] Kevin Pu, Ting Zhang, Naveen Sendhilnathan, Sebastian Freitag, Raj Sodhi, and Tanya R. Jonker. ProMemAssist: Exploring timely proactive assistance through working memory modeling in multi-modal wearable devices. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology*, pages 1–19. ACM, 2025. URL https://doi.org/10.1145/3746059.3747770.

[10] Xiaoan Liu, DaeHo Lee, Eric J Gonzalez, Mar Gonzalez-Franco, and Ryo Suzuki. VisionClaw: Always-on AI agents through smart glasses, 2026. URL https://arxiv.org/abs/2604.03486.

[11] Eric Horvitz, Jack Breese, David Heckerman, David Hovel, and Koos Rommelse. The Lumière project: Bayesian user modeling for inferring the goals and needs of software users. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 1998. URL https://www.microsoft.com/en-us/research/publication/lumiere-project-bayesian-user-modeling-inferring-goals-needs-software-users/.

[12] Yaxi Lu, Shenzhi Yang, Cheng Qian, Guirong Chen, Qinyu Luo, Yesai Wu, Huadong Wang, Xin Cong, Zhong Zhang, Yankai Lin, Weiwen Liu, Yasheng Wang, Zhiyuan Liu, Fangming Liu, and Maosong Sun. Proactive agent: Shifting LLM agents from reactive responses to active assistance, 2024. URL https://arxiv.org/abs/2410.12361.

[13] Nadine Kuo, Agnia Sergeyuk, Valerie Chen, and Maliheh Izadi. Developer interaction patterns with proactive AI: A five-day field study, 2026. URL https://arxiv.org/abs/2601.10253.

[14] Amog Rao, Utkarsh Agarwal, Amol Harsh, and Siddharth Siddharth. AwareLLM: A proactive multimodal ecosystem for personalized Human-AI collaboration to enhance productivity, 2026. URL https://arxiv.org/abs/2605.09625.

[15] Meta. Introducing Muse: The world's first personal AI agent built for everyone, September 2026. URL https://about.fb.com/news/2026/09/introducing-muse-personal-ai-agent/. Product blog.

[16] OpenAI. Introducing dots, September 2026. URL https://openai.com/index/introducing-dots/. Product blog.

[17] Anthropic. Introducing Claude Duet, 2026. URL https://claude.com/resources/articles/claude-duet. Product blog.

[18] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory, 2023. URL https://arxiv.org/abs/2305.10250.

[19] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems, 2024. URL https://arxiv.org/abs/2310.08560.

[20] Yu Wang and Xi Chen. MIRIX: Multi-agent memory system for LLM-Based agents, 2025. URL https://arxiv.org/abs/2507.07957.

[21] Haibo Wang, Bo Feng, Zhengfeng Lai, Mingze Xu, Shiyu Li, Weifeng Ge, Afshin Dehghan, Meng Cao, and Ping Huang. StreamBridge: Turning your offline video large language model into a proactive streaming assistant, 2025. URL https://arxiv.org/abs/2505.05467.

[22] Ao Li, Zihan Xiao, Zihao Yue, Boshen Xu, Linli Yao, Jiaze Li, Pei Fu, Jianzhong Ju, Jian Luan, and Qin Jin. StreamPro: From reactive perception to proactive decision-making in streaming video, 2026. URL https://arxiv.org/abs/2605.16381.

[23] Zexue He, Yu Wang, Churan Zhi, Yuanzhe Hu, Tzu-Ping Chen, Lang Yin, Ze Chen, Tong Arthur Wu, Siru Ouyang, Zihan Wang, Jiaxin Pei, Julian McAuley, Yejin Choi, and Alex Pentland. MemoryArena: Benchmarking agent memory in interdependent multi-session agentic tasks, 2026. URL https://arxiv.org/abs/2602.16313.

[24] Shuhan Xue, Zixin Ding, Yichen Shen, Yinjie Wang, Zhenfei Yin, Yingcheng Wu, Yuxin Chen, Mengdi Wang, and Ling Yang. PAST-Bench: Benchmarking the foundations of recursive self-improvement in personal agents, 2026. URL https://arxiv.org/abs/2608.04003.

[25] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, et al. Tool learning with foundation models. *ACM Computing Surveys*, 57(4), 2024.

[26] Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. Large language model-brained GUI agents: A survey, 2025. URL https://arxiv.org/abs/2411.18279.

[27] Lingdong Kong, Xian Sun, Wei Chow, Linfeng Li, Kevin Qinghong Lin, Xuan Billy Zhang, Song Wang, Rong Li, Qing Wu, Wei Gao, Yingshuo Wang, Shaoyuan Xie, Jiachen Liu, Leigang Qu, Shijie Li, Lai Xing Ng, Benoit R. Cottereau, Ziwei Liu, Tat-Seng Chua, and Wei Tsang Ooi. AI for auto-research: Roadmap & user guide, 2026. URL https://arxiv.org/abs/2605.18661.

[28] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), 2024.

[29] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. The rise and potential of large language model based agents: A survey, 2023. URL https://arxiv.org/abs/2309.07864.

[30] Zane Durante, Qiuyuan Huang, Naoki Wake, Ran Gong, Jae Sung Park, Bidipta Sarkar, Rohan Taori, Yusuke Noda, Demetri Terzopoulos, Yejin Choi, Katsushi Ikeuchi, Hoi Vo, Li Fei-Fei, and Jianfeng Gao. Agent AI: Surveying the horizons of multimodal interaction, 2024. URL https://arxiv.org/abs/2401.03568.

[31] Jiafei Duan, Samson Yu, Hui Li Tan, Hongyuan Zhu, and Cheston Tan. A survey of embodied AI: From simulators to research tasks. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6(2):230–244, 2022.

[32] Meng Chu, Xuan Billy Zhang, Kevin Qinghong Lin, Lingdong Kong, Jize Zhang, Teng Tu, Weijian Ma, Ziqi Huang, Senqiao Yang, Wei Huang, Yeying Jin, Zhefan Rao, Jinhui Ye, Xinyu Lin, Xichen Zhang, Qisheng Hu, Shuai Yang, Leyang Shen, Wei Chow, Yifei Dong, Fengyi Wu, Quanyu Long, Bin Xia, Shaozuo Yu, Mingkang Zhu, Wenhu Zhang, Jiehui Huang, Haokun Gui, Runyi Li, Chenyu Tang, Dong Huang, Xuhang Chen, Rui Liu, Chengzu Li, Shiyi Du, Xu Huang, Haoxuan Che, Long Chen, Qifeng Chen, Wenya Wang, Wenxuan Zhang, Xiaojuan Qi, Yang Deng, Yanwei Li, Mike Zheng Shou, Zhi-Qi Cheng, See-Kiong Ng, Ziwei Liu, Philip Torr, and Jiaya Jia. Agentic world modeling: Foundations, capabilities, laws, and beyond, 2026. URL https://arxiv.org/abs/2604.22748.

[33] Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, Zhenrong Cheng, Xuanbo Fan, Jiaxin Guo, Xinlei Yu, Zhenhong Zhou, Zewen Hu, Jiahao Huo, Junhao Wang, Yuwei Niu, Yu Wang, Zhenfei Yin, Xiaobin Hu, Yue Liao, Qiankun Li, Kun Wang, Wangchunshu Zhou, Yixin Liu, Dawei Cheng, Qi Zhang, Tao Gui, Shirui Pan, Yan Zhang, Philip Torr, Zhicheng Dou, Ji-Rong Wen, Xuanjing Huang, Yu-Gang Jiang, and Shuicheng Yan. Memory in the age of AI agents, 2026. URL https://arxiv.org/abs/2512.13564.

[34] Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model based agents, 2024. URL https://arxiv.org/abs/2404.13501.

[35] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey, 2024. URL https://arxiv.org/abs/2312.10997.

[36] Yue Xu, Qian Chen, Zizhan Ma, Dongrui Liu, Wenxuan Wang, Xiting Wang, Li Xiong, and Wenjie Wang. Toward personalized LLM-Powered agents: Foundations, evaluation, and future directions, 2026. URL https://arxiv.org/abs/2602.22680.

[37] Zehao Lin, Xixuan Hao, Renyu Fu, Shaobo Cui, Kai Chen, Chunyu Li, Zhiyu Li, and Feiyu Xiong. A survey on long-term memory security in LLM agents: Attacks, defenses, and governance across the memory lifecycle, 2026. URL https://arxiv.org/abs/2604.16548.

[38] Yang Deng, Wenqiang Lei, Wai Lam, and Tat-Seng Chua. A survey on proactive dialogue systems: Problems, methods, and prospects. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, 2023. URL https://arxiv.org/abs/2305.02750.

[39] Yang Deng, Lizi Liao, Zhonghua Zheng, Grace Hui Yang, and Tat-Seng Chua. Towards human-centered proactive conversational agents. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2024. URL https://arxiv.org/abs/2404.12670.

[40] Yang Deng, Lizi Liao, Wenqiang Lei, Grace Hui Yang, Wai Lam, and Tat-Seng Chua. Proactive conversational AI: A comprehensive survey of advancements and opportunities. *ACM Transactions on Information Systems*, 43(3), 2025.

[41] Bingbing Wen, Jihan Yao, Shangbin Feng, Chenjun Xu, Yulia Tsvetkov, Bill Howe, and Lucy Lu Wang. Know your limits: A survey of abstention in large language models, 2025. URL https://arxiv.org/abs/2407.18418.

[42] Huan-ang Gao et al. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *Transactions on Machine Learning Research*, 2026. URL https://openreview.net/forum?id=CTr3bovS5F.

[43] Zhe Ren, Yimeng Chen, Dandan Guo, Guowei Rong, Tonghui Li, R. B. Xiong, Qingfeng Lan, Wenyi Wang, Li Nanbo, Yibo Yang, Mingchen Zhuge, and Jürgen Schmidhuber. Self-improvements in modern agentic systems: A survey, 2026. URL https://arxiv.org/abs/2607.13104.

[44] Yubo Li. Dynamic agent skills: A lifecycle survey and taxonomy of evolving skill libraries, 2026. URL https://arxiv.org/abs/2607.10113.

[45] Jiajun Xu, Zhiyuan Li, Wei Chen, Qun Wang, Xin Gao, Qi Cai, and Ziyuan Ling. On-device language models: A comprehensive review, 2024. URL https://arxiv.org/abs/2409.00088.

[46] Cathal Gurrin, Alan F. Smeaton, and Aiden R. Doherty. LifeLogging: Personal big data. *Foundations and Trends in Information Retrieval*, 8(1):1–125, 2014.

[47] Sizhen Bian, Mengxi Liu, Lala Shakti Swarup Ray, Bo Zhou, Bin Guo, Zhiwen Yu, Thomas Ploetz, Paul Lukowicz, Siyu Yuan, and Vitor Fortes Rey. Foundation models defining a new era in sensor-based human activity recognition: A survey and outlook, 2026. URL https://arxiv.org/abs/2604.02711.

[48] Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. A survey on multimodal large language models. *National Science Review*, 11(12), 2024.

[49] Yolo Y. Tang, Jing Bi, Siting Xu, Luchuan Song, Susan Liang, Teng Wang, Daoan Zhang, Jie An, Jingyang Lin, Rongyi Zhu, Ali Vosoughi, Chao Huang, Zeliang Zhang, Pinxin Liu, Mingqian Feng, Feng Zheng, Jianguo Zhang, Ping Luo, Jiebo Luo, and Chenliang Xu. Video understanding with large language models: A survey, 2025. URL https://arxiv.org/abs/2312.17432.

[50] You Qin, Kai Liu, Shengqiong Wu, Kai Wang, Shijian Deng, Yapeng Tian, Junbin Xiao, Yazhou Xing, Yinghao Ma, Bobo Li, Roger Zimmermann, Lei Cui, Furu Wei, Jiebo Luo, and Hao Fei. Audio-visual intelligence in large foundation models, 2026. URL https://arxiv.org/abs/2605.04045.

[51] Xiang Li, Heqian Qiu, Lanxiao Wang, Hanwen Zhang, Chenghao Qi, Linfeng Han, Huiyu Xiong, and Hongliang Li. Challenges and trends in egocentric vision: A survey, 2025. URL https://arxiv.org/abs/2503.15275.

[52] Chiara Plizzari, Gabriele Goletto, Antonino Furnari, Siddhant Bansal, Francesco Ragusa, Giovanni Maria Farinella, Dima Damen, and Tatiana Tommasi. An outlook into the future of egocentric vision. *International Journal of Computer Vision*, 2024.

[53] Junlong Tong, Zilong Wang, YuJie Ren, Peiran Yin, Hao Wu, Wei Zhang, and Xiaoyu Shen. From static inference to dynamic interaction: A survey of streaming large language models, 2026. URL https://arxiv.org/abs/2603.04592.

[54] Yan Tang, Tingyu Cao, Yuanbo Tang, Huaze Tang, and Keer Hu. Proactive service agents: A unified decision framework, methods, and evaluation, 2026. URL https://arxiv.org/abs/2609.03727.

[55] Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, Zhaochun Ren, Nikos Aletras, Xi Wang, Han Zhou, and Zaiqiao Meng. A comprehensive survey of self-evolving AI agents: A new paradigm bridging foundation models and lifelong agentic systems, 2025. URL https://arxiv.org/abs/2508.07407.

[56] Kaiqi Wang, Wenjin Hou, Yuchen Yan, Hongrui Jia, Zhisheng Zhong, Botao Ren, Yifei Chen, Songyang Zhang, Yongliang Shen, Jun Xiao, Yi Yang, Yueting Zhuang, and Hehe Fan. Diving into reliable self-evolving agents: A survey. *Preprints.org preprint 202609.0913*, 2026. doi: 10.20944/preprints202609.0913.v1. URL https://doi.org/10.20944/preprints202609.0913.v1.

[57] Qianggang Ding, Xingyao Wang, Rui Feng, Zhibin Wang, Feixiang Yao, Kelong Mao, Hao Sun, Zhiyao Luo, Jiankai Tang, Lei Li, Jiadong Guo, Minheng Ni, Weicong Lin, Chenxi Yang, Hongxiang Gao, Zhenghua Chen, Yang Bai, Min Wu, Jun Cheng, Huazhu Fu, Dacheng Tao, and Bang Liu. ComBodied Agents: a new paradigm of human-centric agentic AI, 2026. URL https://arxiv.org/abs/2608.10915.

[58] Xu Yuan, Yi Wang, Zhuohang Jiang, Haohao Qu, Yujuan Ding, Shanru Lin, Guoliang Xing, Hongxia Yang, Jiannong Cao, Qing Li, and Wenqi Fan. AI smart glasses for wearable intelligence: From egocentric sensing to agentic personalization, 2026. URL https://arxiv.org/abs/2609.19793.

[59] Jiangning Zhang, Haojun Chen, and Yong Liu. From seeing to acting: Smart glasses as first-person intelligence platforms, 2026. URL https://arxiv.org/abs/2608.24877.

[60] Guanting Dong, Xiaoshuai Song, Yuyang Hu, Jiajie Jin, Chenghao Zhang, Yifei Chen, Xiaoxi Li, Huaying Yuan, Xinyu Yang, Tongyu Wen, Jiejun Tan, Hongjin Qian, Shijue Huang, Junting Lu, Zhenyu Li, Wanjun Zhong, Yutao Zhu, Tat-Seng Chua, Zhicheng Dou, and Ji-Rong Wen. Towards Long-Horizon Agents: A Survey. *Preprints*, July 2026. doi: 10.20944/preprints202607.1328.v1. URL https://doi.org/10.20944/preprints202607.1328.v1.

[61] Ke Yang, Jiateng Liu, Jize Jiang, Chun Zhao, Yufan Cao, Shanshan Zhong, Yiyang Du, Ruixuan Liu, Xuying Ning, Dean Alvarez, Yuji Zhang, Duo Zhou, Miri Liu, Yuhao Cheng, Hanyang Chen, Rui Yang, Zifeng Wang, Hanghang Tong, Jingrui He, Chenyan Xiong, Chen-Yu Lee, Jiawei Han, and ChengXiang Zhai. Proactive AI for superintelligent agents, 2026. URL https://empathyang.github.io/Proactive-AI-for-Superintelligent-Agents/.

[62] Tianyu Ding, Aditya Nannapaneni, Bingfan Liu, and Ling Zhang. Always-on agents: A survey of persistent memory, state, and governance in LLM agents, 2026. URL https://arxiv.org/abs/2606.30306.

[63] Inbal Nahum-Shani, Shawna N Smith, Bonnie J Spring, Linda M Collins, Katie Witkiewitz, Ambuj Tewari, and Susan A Murphy. Just-in-time adaptive interventions (JITAIs) in mobile health: key components and design principles for ongoing health behavior support. *Annals of Behavioral Medicine*, 52(6):446–462, 2018. doi: 10.1007/s12160-016-9830-8.

[64] Peng Liao, Zhengling Qi, Runzhe Wan, Predrag Klasnja, and Susan Murphy. Batch policy learning in average reward Markov decision processes, 2022. URL https://arxiv.org/abs/2007.11771.

[65] Archit Sharma, Kelvin Xu, Nikhil Sardana, Abhishek Gupta, Karol Hausman, Sergey Levine, and Chelsea Finn. Autonomous reinforcement learning: Formalism and benchmarking. *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2112.09605.

[66] Lixin Zou, Long Xia, Zhuoye Ding, Jiaxing Song, Weidong Liu, and Dawei Yin. Reinforcement learning to optimize long-term user engagement in recommender systems, 2019. URL https://arxiv.org/abs/1902.05570.

[67] Qingpeng Cai, Shuchang Liu, Xueliang Wang, Tianyou Zuo, Wentao Xie, Bin Yang, Dong Zheng, Peng Jiang, and Kun Gai. Reinforcing user retention in a billion scale short video recommender system, 2023. URL https://arxiv.org/abs/2302.01724.

[68] John E. Laird. Introduction to Soar, 2022. URL https://arxiv.org/abs/2205.03854.

[69] Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024. URL https://arxiv.org/abs/2309.02427.

[70] Ruzena Bajcsy. Active perception. *Proceedings of the IEEE*, 76(8):966–1005, 1988.

[71] Kenneth J. W. Craik. *The Nature of Explanation*. Cambridge University Press, 1943.

[72] Yann LeCun. A path towards autonomous machine intelligence. OpenReview preprint, version 0.9.2, 2022. URL https://openreview.net/forum?id=BZ5a1r-kVsf.

[73] Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, Chenlin Zhou, Jiayi Mao, Tianze Xia, Jiafeng Guo, and Shenghua Liu. A survey of context engineering for large language models, 2025. URL https://arxiv.org/abs/2507.13334.

[74] Joya Chen, Zhaoyang Lv, Shiwei Wu, Kevin Qinghong Lin, Chenan Song, Difei Gao, Jia-Wei Liu, Ziteng Gao, Dongxing Mao, and Mike Zheng Shou. VideoLLM-online: Online video large language model for streaming video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18407–18418, 2024. URL https://arxiv.org/abs/2406.11816.

[75] Yi Duan, Ying Liu, Zirui Tang, Haodong Chen, Jun Zhou, Yumou Liu, Bangrui Xu, Yukai Wu, Sidi Chen, Yuhan Zhou, Haoyu Wang, Xiaoyou Yu, Shaokun Han, Xuzhou Zhu, Le Zhou, Bolin Lu, Wei Zhou, Jiachen Liu, Nuozhou Fang, Jiaxin Tian, Ruoyu Chen, Yuxuan Li, Kai Zuo, Kaiyan Zhang, Qianyu Yang, Zijie Wang, Jiantao Qiu, Conghui He, Guoliang Li, Bowen Zhou, Zhiyuan Liu, Zhoufutu Wen, Jihua Kang, Xuanhe Zhou, and Fan Wu. The last AI built by humans: Toward genuine recursive self-improvement, 2026. URL https://arxiv.org/abs/2609.11873.

[76] Gregory D. Abowd, Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper, and Mike Pinkerton. Cyberguide: A mobile context-aware tour guide. *Wireless Networks*, 3(5):421–433, 1997. doi: 10.1023/A:1019194325861. URL https://sites.cc.gatech.edu/fce/cyberguide/index.html.

[77] Jennifer Healey and Rosalind W. Picard. StartleCam: A cybernetic wearable camera. In *Proceedings of the Second International Symposium on Wearable Computers*, 1998. URL https://vismod.media.mit.edu/pub/tech-reports/TR-468/.

[78] Steve Hodges, Lyndsay Williams, Emma Berry, Shahram Izadi, James Srinivasan, Alex Butler, Gavin Smyth, Narinder Kapur, and Ken Wood. SenseCam: A retrospective memory aid. In *Proceedings of the 8th International Conference on Ubiquitous Computing (UbiComp)*, pages 177–193, 2006.

[79] Vannevar Bush. As we may think. *The Atlantic Monthly*, 176(1):101–108, 1945.

[80] Bradley J. Rhodes and Thad Starner. Remembrance agent: A continuously running automated information retrieval system. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM)*, pages 487–495, 1996.

[81] Susan Dumais, Edward Cutrell, J. J. Cadiz, Gavin Jancke, Raman Sarin, and Daniel Robbins. Stuff I've Seen: A system for personal information retrieval and re-use. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2003. URL https://www.microsoft.com/en-us/research/publication/stuff-ive-seen-a-system-for-personal-information-retrieval-and-re-use/.

[82] Karen Myers, Pauline Berry, Jim Blythe, Ken Conley, Melinda Gervasio, Deborah L. McGuinness, David Morley, Avi Pfeffer, Martha Pollack, and Milind Tambe. An intelligent personal assistant for task and time management. *AI Magazine*, 28(2):47–61, 2007.

[83] Eric Horvitz, Paul Koch, and Johnson Apacible. BusyBody: Creating and fielding personalized models of the cost of interruption. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, 2004. URL https://www.microsoft.com/en-us/research/publication/busybody-creating-fielding-personalized-models-cost-interruption/.

[84] Kenton Lee, Mandar Joshi, Iulia Turc, Hexiang Hu, Fangyu Liu, Julian Eisenschlos, Urvashi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. Pix2Struct: Screenshot parsing as pretraining for visual language understanding, 2023. URL https://arxiv.org/abs/2210.03347.

[85] Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing GUI grounding for advanced visual GUI agents, 2024. URL https://arxiv.org/abs/2401.10935.

[86] Gilles Baechler, Srinivas Sunkara, Maria Wang, Fedir Zubach, Hassan Mansoor, Vincent Etter, Victor Cărbune, Jason Lin, Jindong Chen, and Abhanshu Sharma. ScreenAI: A vision-language model for UI and infographics understanding, 2024. URL https://arxiv.org/abs/2402.04615.

[87] Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. OmniParser for pure vision based GUI agent, 2024. URL https://arxiv.org/abs/2408.00203.

[88] Ishan Chatterjee, Maruchi Kim, Vivek Jayaram, Shyamnath Gollakota, Ira Kemelmacher-Shlizerman, Shwetak Patel, and Steven M. Seitz. ClearBuds: Wireless binaural earbuds for learning-based speech enhancement. In *ACM MobiSys*, 2022. doi: 10.1145/3498361.3538933.

[89] Bandhav Veluri, Malek Itani, Justin Chan, Takuya Yoshioka, and Shyamnath Gollakota. Semantic hearing: Programming acoustic scenes with binaural hearables, 2023. URL https://arxiv.org/abs/2311.00320.

[90] Erik Ekstedt and Gabriel Skantze. Voice activity projection: Self-supervised learning of turn-taking events. In *Interspeech*, 2022. doi: 10.21437/Interspeech.2022-10955.

[91] Alexandre Défossez, Laurent Mazaré, Manu Orsini, Amélie Royer, Patrick Pérez, Hervé Jégou, Edouard Grave, and Neil Zeghidour. Moshi: a speech-text foundation model for real-time dialogue, 2024. URL https://arxiv.org/abs/2410.00037.

[92] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic memory for LLM agents, 2025. URL https://arxiv.org/abs/2502.12110.

[93] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024. URL https://arxiv.org/abs/2404.07972.

[94] Jingkang Yang, Shuai Liu, Hongming Guo, Yuhao Dong, Xiamengwei Zhang, Sicheng Zhang, Pengyun Wang, Zitang Zhou, Binzhu Xie, Ziyue Wang, et al. Egolife: Towards egocentric life assistant. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 28885–28900, 2025.

[95] Lin Long, Yichen He, Wentao Ye, Yiyuan Pan, Yuan Lin, Hang Li, Junbo Zhao, and Wei Li. Seeing, listening, remembering, and reasoning: A multimodal agent with long-term memory, 2025. URL https://arxiv.org/abs/2508.09736.

[96] Bufang Yang, Lilin Xu, Liekang Zeng, Kaiwei Liu, Siyang Jiang, Wenrui Lu, Hongkai Chen, Xiaofan Jiang, Guoliang Xing, and Zhenyu Yan. ContextAgent: Context-aware proactive LLM agents with open-world sensory perceptions, 2025. URL https://arxiv.org/abs/2505.14668.

[97] Figure. Helix: A vision-language-action model for generalist humanoid control, February 2025. URL https://www.figure.ai/news/helix. Product blog.

[98] Qianshu Cai, Yonggang Zhang, Xianzhang Jia, Huajiang Zheng, Wei Xue, Jun Song, Xinmei Tian, and Yike Guo. MOSS: Self-evolution through source-level rewriting in autonomous agent systems, 2026. URL https://arxiv.org/abs/2605.22794.

[99] Anton Razzhigaev, Andrei Gritsaev, Andrei Kaznacheev, Nikita Dragunov, Roman Yampolskiy, and Andrei Kuznetsov. Ouroboros: A self-developing frontier coding agent with reviewed core evolution, 2026. URL https://arxiv.org/abs/2608.08311.

[100] Intel. New Intel RealSense D435i Stereo Depth Camera Adds 6 Degrees of Freedom Tracking, 2018. URL https://download.intel.com/newsroom/2021/archive/2018-11-13-news-new-intel-realsense-d435i-stereo-depth-camera-adds-6-degrees-freedom-tracking.pdf. Product blog.

[101] Bosch Sensortec. CES 2018: Bosch Launches BMA400 Ultra-Low Power Accelerometer, 2018. URL https://www.bosch-sensortec.com/en/news/ces-2018-bosch-launches-bma400-ultra-low-power-accelerometer.html. Product blog.

[102] Vuzix. M400 Smart Glasses, 2026. URL https://www.vuzix.com/products/m400-smart-glasses. Product blog.

[103] rabbit inc. Introducing r1, a Pocket Companion That Moves AI from Words to Action, 2024. URL https://www.rabbit.tech/newsroom/introducing-r1. Product blog.

[104] Samsung Electronics. Introducing Galaxy XR: Opening New Worlds, 2025. URL https://news.samsung.com/global/introducing-galaxy-xr-opening-new-worlds. Product blog.

[105] Meta. Inside Aria Gen 2: Explore the Cutting-Edge Tech Behind the Device, 2025. URL https://ai.meta.com/blog/aria-gen-2-research-glasses-under-the-hood-reality-labs/. Product blog.

[106] Figure. Introducing Figure 03, 2025. URL https://www.figure.ai/news/introducing-figure-03. Product blog.

[107] X Square Robot. Quanta X2 and WALL-A: Company Development Milestones, 2026. URL https://x2robot.com/en/about. Product blog.

[108] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 4th edition, 2020.

[109] Baoqing Yue, Zihan Zhu, Yutong Han, Brian Fan, Qian Sun, Jichen Feng, Hufei Yang, Yifan Zhang, and Mengdi Wang. Interactive benchmarks, 2026. URL https://arxiv.org/abs/2603.04737.

[110] Sechang Oh et al. An acoustic signal processing chip with 142-nW voice activity detection using mixer-based sequential frequency scanning and neural network classification. *IEEE Journal of Solid-State Circuits*, 54 (11), 2019. doi: 10.1109/JSSC.2019.2936756.

[111] Juan Sebastian P. Giraldo, Steven Lauwereins, Komail Badami, and Marian Verhelst. Vocell: A 65-nm speech-triggered wake-up SoC for 10-μw keyword spotting and speaker verification. *IEEE Journal of Solid-State Circuits*, 55(4):868–878, 2020. doi: 10.1109/JSSC.2020.2968800.

[112] Shaojin Ding, Quan Wang, Shuo yiin Chang, Li Wan, and Ignacio Lopez Moreno. Personal VAD: Speaker-conditioned voice activity detection, 2026. URL https://arxiv.org/abs/1908.04284.

[113] Akhil Padmanabha, Saravanan Govindarajan, Hwanmun Kim, Sergio Ortiz, Rahul Rajan, Doruk Senkal, and Sneha Kadetotad. EgoCHARM: Resource-efficient hierarchical activity recognition using an egocentric IMU sensor, 2025. URL https://arxiv.org/abs/2504.17735.

[114] Jiacheng Liu, Yuanchun Li, Liangyan Li, Yi Sun, Hao Wen, Xiangyu Li, Yao Guo, and Yunxin Liu. ChainStream: An LLM-based framework for unified synthetic sensing, 2024. URL https://arxiv.org/abs/2412.15240.

[115] Nossa Iyamu. Activity frames: Deterministic screen-activity compilation for agent memory and replay, 2026. URL https://arxiv.org/abs/2608.05784.

[116] Jim Gemmell, Gordon Bell, and Roger Lueder. MyLifeBits: A personal database for everything. *Communications of the ACM*, 49(1):88–95, 2006.

[117] Ruyi Xu, Guangxuan Xiao, Yukang Chen, Liuning He, Yao Lu, and Song Han. StreamingVLM: Real-Time Understanding for Infinite Video Streams, 2026. URL https://arxiv.org/abs/2510.09608.

[118] Senqiao Yang, Kaichen Zhang, Zhaoyang Jia, Jinghao Guo, Yifei Shen, Xinjie Zhang, Xiaoyi Zhang, Haoqing Wang, Xiao Li, Peng Zhang, Xiang An, Yin Xie, Zhening Liu, Xun Guo, Jiahao Li, Shicheng Zheng, Jinglu Wang, Zongyu Guo, Wenxuan Xie, Zihan Zheng, Yuxuan Luo, Bin Li, and Yan Lu. Mage-VL: An efficient codec-native streaming multimodal foundation model, 2026. URL https://arxiv.org/abs/2607.24904.

[119] Yujie Tu, Zhiliang Peng, Jianwei Yu, Li Dong, Songchen Xu, Yaoyao Chang, Wenhui Wang, Zilong Wang, Zehua Wang, Yan Xia, Jiajun Zhang, Xie Chen, and Furu Wei. VibeVoice-ASR-Streaming Technical Report, 2026. URL https://arxiv.org/abs/2609.02812.

[120] Meta. Project Aria Docs, 2026. URL https://facebookresearch.github.io/projectaria_tools/gen2/. Product blog.

[121] Akshay Paruchuri, Sinan Hersek, Lavisha Aggarwal, Qiao Yang, Xin Liu, Achin Kulshrestha, Andrea Colaco, Henry Fuchs, and Ishan Chatterjee. EgoTrigger: Toward audio-driven image capture for human memory enhancement in all-day energy-efficient smart glasses. *IEEE Transactions on Visualization and Computer Graphics*, 2025.

[122] Pietro Bonazzi, Julian Moosmann, Ahmet Celik, Philipp Mayer, and Michele Magno. OpenGlass: Ultra-low-power on-device AI eyewear with event-based vision, 2026. URL https://arxiv.org/abs/2606.07431.

[123] Masoud Jalayer, Changyi Li, and Yu Xiao. Pre-decoding acoustic triage for budgeted vision-language captioning of untrimmed egocentric video, 2026. URL https://arxiv.org/abs/2608.22359.

[124] Zhijie Ding, Weinan Hong, Zicheng Zhu, Lei Li, Dezhi Kong, Hao Wang, Peng Zhou, Xuchu Jiang, and Jiaming Xu. Perceive before reasoning: A pre-reasoning perception framework for efficient and reliable proactive mobile agents, 2026. URL https://arxiv.org/abs/2606.03236.

[125] Xin Ding, Hao Wu, Yifan Yang, Shiqi Jiang, Qianxi Zhang, Donglin Bai, Zhibo Chen, and Ting Cao. Streammind: Unlocking full frame rate streaming video dialogue through event-gated cognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13448–13459, 2025.

[126] Zhenghui Guo, Yuanbin Man, Junyuan Sheng, Bowen Lin, Ahmed Ahmed, Bo Jiang, Boyuan Zhang, Miao Yin, Sian Jin, Omprakash Gnawal, and Chengming Zhang. Event-VStream: Event-driven real-time understanding for long video streams, 2026. URL https://arxiv.org/abs/2601.15655.

[127] Shusheng Yang, Jihan Yang, Pinzhi Huang, Ellis Brown, Zihao Yang, Yue Yu, Shengbang Tong, Zihan Zheng, Yifan Xu, Muhan Wang, Rob Fergus, Yann LeCun, Li Fei-Fei, and Saining Xie. Cambrian-S: Towards spatial supersensing in video. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2511.04670.

[128] Yikai Zheng, Xin Ding, Yifan Yang, Shiqi Jiang, Hao Wu, Qianxi Zhang, Weijun Wang, Ting Cao, and Yunxin Liu. Em-garde: A propose-match framework for proactive streaming video understanding, 2026. URL https://arxiv.org/abs/2603.19054.

[129] Jinzhao Li, Yinuo Chen, Wenxuan Song, Yijia Lei, Yichi Zhang, Honglei Yan, Panwang Pan, and Miao Liu. IPIBench: Evaluating interactive proactive intelligence of MLLMs under continuous streams, 2026. URL https://arxiv.org/abs/2605.27074.

[130] Bufang Yang, Lilin Xu, Liekang Zeng, Yunqi Guo, Siyang Jiang, Wenrui Lu, Kaiwei Liu, Yixuan Li, Xiaofan Jiang, Guoliang Xing, and Zhenyu Yan. ProAgent: Harnessing on-demand sensory contexts for proactive LLM agent systems in the wild, 2026. URL https://arxiv.org/abs/2512.06721.

[131] Zichen Wen, Yiyu Wang, Chenfei Liao, Boxue Yang, Junxian Li, Weifeng Liu, Haocong He, Bolong Feng, Xuyang Liu, Yuanhuiyi Lyu, Xu Zheng, Xuming Hu, and Linfeng Zhang. AI for service: Proactive assistance with AI glasses, 2025. URL https://arxiv.org/abs/2510.14359.

[132] Zhiyuan Wang, Subigya Nepal, Ariful Islam, Indrajeet Ghosh, Xinyu Chen, Katharine E. Daniel, Laura E. Barnes, and Philip Chow. PULSE: Agentic investigation with passive sensing for proactive affective intervention in cancer survivorship, 2026. URL https://arxiv.org/abs/2605.17679.

[133] Dmitriy Rivkin, Francois Hogan, Amal Feriani, Abhisek Konar, Adam Sigal, Steve Liu, and Greg Dudek. SAGE: Smart home agent with grounded execution, 2024. URL https://arxiv.org/abs/2311.00772.

[134] Yoonsang Kim, Devshree Jadeja, Divyansh Pradhan, Yalong Yang, and Arie Kaufman. SpeechLess: Micro-utterance with personalized spatial memory-aware assistant in everyday augmented reality, 2026. URL https://arxiv.org/abs/2602.00793.

[135] Riku Arakawa, Prasoon Patidar, Will Page, Jill Lehman, and Mayank Goel. Scaling context-aware task assistants that learn from demonstration and adapt through mixed-initiative dialogue. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology*, pages 1–19, 2025. doi: 10.1145/3746059.3747700. URL https://doi.org/10.1145/3746059.3747700.

[136] Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. Mobile-Agent-E: Self-Evolving Mobile Assistant for Complex Tasks, 2025. URL https://arxiv.org/abs/2501.11733.

[137] Yijun Chen, Boyi Xiao, Yixian Zhao, Haoting Xia, Buqiang Xu, Jizhan Fang, Yanya Li, Yaqi Zheng, Xuehai Wang, Zirui Xue, Liuxin Zhang, Hui Li, and Ningyu Zhang. LightMem-Ego: Your AI Memory for Everyday Life, 2026. URL https://arxiv.org/abs/2607.11487.

[138] Based Hardware. Omi: Wearable AI notetaker, 2026. URL https://www.omi.me/. Product blog.

[139] Microsoft. Update on the Recall preview feature for Copilot+ PCs, 2024. URL https://blogs.windows.com/windowsexperience/2024/06/07/update-on-the-recall-preview-feature-for-copilot-pcs/. Product blog.

[140] Sudershan Boovaraghavan et al. Kirigami: Lightweight speech filtering for privacy-preserving activity recognition using audio. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 8(1), 2024. doi: 10.1145/3643502.

[141] NeoteAI Team and Fudan TEAI Team. N_0-TWAM: Scaling Tactile-Native World-Action Model for Contact-Rich Manipulation, 2026. URL https://arxiv.org/abs/2607.23783.

[142] Michele Fiori, Gabriele Civitarese, Flora D. Salim, and Claudio Bettini. DomusFM: A foundation model for event-based behavioral monitoring in smart-homes, 2026. URL https://arxiv.org/abs/2602.01910.

[143] Abrar Anwar, John Welsh, Joydeep Biswas, Soha Pouya, and Yan Chang. ReMEmbR: Building and reasoning over long-horizon spatio-temporal memory for robot navigation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2025. URL https://arxiv.org/abs/2409.13682.

[144] Rui Qian, Shuangrui Ding, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Yuhang Cao, Dahua Lin, and Jiaqi Wang. Dispider: Enabling video llms with active real-time interaction via disentangled perception, decision, and reaction. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 24045–24055, 2025.

[145] Lilin Xu, Bufang Yang, Siyang Jiang, Kaiwei Liu, Kaiyuan Hou, Yuang Fan, Hongkai Chen, Zhenyu Yan, and Xiaofan Jiang. Pro^{2}Assist: Continuous step-aware proactive assistance with multi-modal egocentric perception for long-horizon procedural tasks, 2026. URL https://arxiv.org/abs/2605.04227.

[146] Yuhong Zhang, Khanh An Nguyen, Xinyu Tian, KungYi Chang, Abraham Padua, Yuchen Xu, Ying Choon Wu, Gert Cauwenberghs, and Tzyy-Ping Jung. GazeAgent: VLM-guided implicit gaze calibration from egocentric multimodal behavior for AI glasses, 2026. URL https://arxiv.org/abs/2610.24364.

[147] Xiaoze Liu, Ruowang Zhang, Amir H. Abdi, Michel Galley, Zhikai Chen, Siheng Xiong, Xiaoqian Wang, and Jing Gao. Do proactive agents need an LLM to decide when to act?, 2026. URL https://arxiv.org/abs/2605.30152.

[148] James F. Mullen, Dhruva Kumar, Xuewei Qi, Rajasimman Madhivanan, Arnie Sen, Dinesh Manocha, and Richard Kim. HomeEmergency: Using audio to find and respond to emergencies in the home. *IEEE Robotics and Automation Letters*, 2025. URL https://arxiv.org/abs/2504.01089.

[149] Fengkai Liu, Hao Su, Haozhuang Chi, Rui Geng, Congzhi Ren, Xuqing Liu, Chenfei Xu, Yuichi Ohsita, and Liyun Zhang. Proactive robot assistance via event-driven reasoning over state changes, 2026. URL https://arxiv.org/abs/2610.08344.

[150] Enshen Zhou, Qi Su, Cheng Chi, Zhizheng Zhang, Zhongyuan Wang, Tiejun Huang, Lu Sheng, and He Wang. Code-as-monitor: Constraint-aware visual programming for reactive and proactive robotic failure detection, 2025. URL https://arxiv.org/abs/2412.04455.

[151] Sicheng Yang, Yukai Huang, Weitong Cai, Shitong Sun, Fengyi Fang, You He, Yiqiao Xie, Jiankang Deng, Hang Zhang, Jifei Song, and Zhensong Zhang. Egocentric co-pilot: Web-native smart-glasses agents for assistive egocentric AI, 2026. URL https://arxiv.org/abs/2603.01104.

[152] Haoyi Hu, Qirong Lyu, Xianghan Kong, Weiwen Liu, Jianghao Lin, Zixuan Guo, Yan Xu, Yasheng Wang, Weinan Zhang, and Yong Yu. Anticipate and learn: Unleashing idle-time compute in proactive agents, 2026. URL https://arxiv.org/abs/2605.25971.

[153] Yuanda Hu, Wenbin Zuo, Yiting Shen, Tianle Chen, Hector Fabio Calero Tobar, Yate Ge, Xiaohua Sun, and Weiwei Guo. EgoAsk: Egocentric teaching of personalized object knowledge for household robots, 2026. URL https://arxiv.org/abs/2609.16766.

[154] Yuzhe Zhang, Xianwei Xue, Xingyong Wu, Mengke Chen, Chen Liu, Xinran He, Run Shao, Feiran Liu, Huanmin Xu, Qiutong Pan, and Haiwei Wang. Don't act blindly: Robust GUI automation via action-effect verification and self-correction, 2026. URL https://arxiv.org/abs/2604.05477.

[155] Kush Hari, Justin Kerr, Nidhya Shivakumar, Samarth Mahapatra, Carmelo Sferrazza, Jiahui Lei, Jitendra Malik, C. Karen Liu, Ken Goldberg, and Angjoo Kanazawa. EyeRobot 2.0: Active gaze for precise manipulation without wrist cameras, 2026. URL https://arxiv.org/abs/2610.03710.

[156] Haoyu Xiong, Xiaomeng Xu, Jimmy Wu, Yifan Hou, Jeannette Bohg, and Shuran Song. Vision in action: Learning active perception from human demonstrations, 2025. URL https://arxiv.org/abs/2506.15666.

[157] Allen Z. Ren, Jaden Clark, Anushri Dixit, Masha Itkina, Anirudha Majumdar, and Dorsa Sadigh. Explore until confident: Efficient exploration for embodied question answering. In *Robotics: Science and Systems (RSS)*, 2024. URL https://arxiv.org/abs/2403.15941.

[158] Matteo Luperto, Javier Monroy, Jennifer Renoux, Francesca Lunardini, Nicola Basilico, Maria Bulgheroni, Angelo Cangelosi, Matteo Cesari, Manuel Cid, Aladar Ianes, Javier Gonzalez-Jimenez, Anastasis Kounoudes, David Mari, Victor Prisacariu, Arso Savanovic, Simona Ferrante, and N. Alberto Borghese. Integrating social assistive robots, IoT, virtual communities and smart objects to assist at-home independently living elders: the MoveCare project. *International Journal of Social Robotics*, 15(3):517–545, 2023. doi: 10.1007/s12369-021-00843-0.

[159] Noam Shazeer. Fast transformer decoding: One write-head is all you need, 2019. URL https://arxiv.org/abs/1911.02150.

[160] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.17453.

[161] Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Will Tennien, Atri Rudra, James Zou, Azalia Mirhoseini, and Christopher Re. Cartridges: Lightweight and general-purpose long context representations via self-study, 2025. URL https://arxiv.org/abs/2506.06266.

[162] Feilong Tang, Xiang An, Yunyao Yan, Yin Xie, Bin Qin, Kaicheng Yang, Yifei Shen, Yuanhan Zhang, Chunyuan Li, Shikun Feng, Changrui Chen, Huajie Tan, Ming Hu, Manyuan Zhang, Bo Li, Ziyong Feng, Ziwei Liu, Zongyuan Ge, and Jiankang Deng. OneVision-Encoder: Codec-aligned sparsity as a foundational principle for multimodal intelligence, 2026. URL https://arxiv.org/abs/2602.08683.

[163] Xiang An, Yin Xie, Feilong Tang, Yunyao Yan, Huajie Tan, Didi Zhu, Changrui Chen, Xiuwei Zhao, Bin Qin, Kaicheng Yang, Yifei Shen, Yuanhan Zhang, Kaichen Zhang, Wenkang Zhang, Zheng Cheng, Nansen Zhang, Chunsheng Wu, Chunjiang Ge, Zimin Ran, Dehua Song, Chunyuan Li, Shikun Feng, Ming Hu, Zhangquan Chen, Junbo Niu, Bo Li, Ziyong Feng, Ziwei Liu, Zongyuan Ge, and Jiankang Deng. LLaVA-OneVision-2: Towards next-generation perceptual intelligence, 2026. URL https://arxiv.org/abs/2605.25979.

[164] Zijia Zhao, Yuqi Huo, Tongtian Yue, Longteng Guo, Haoyu Lu, Bingning Wang, Weipeng Chen, and Jing Liu. Efficient motion-aware video MLLM, 2025. URL https://arxiv.org/abs/2503.13016.

[165] Sayan Deb Sarkar, Rémi Pautrat, Ondrej Miksik, Marc Pollefeys, Iro Armeni, Mahdi Rad, and Mihai Dusmanu. CoPE-VideoLM: Leveraging codec primitives for efficient video language modeling, 2026. URL https://arxiv.org/abs/2602.13191.

[166] Guijia Zhang, Yuxun Chen, Yuheng Qi, and Harry Yang. Do GUI agents believe their eyes? diagnosing state-belief reliance on pixels versus structure, 2026. URL https://arxiv.org/abs/2607.04334.

[167] Strick Sheng, Ziyue Wang, and Liyi Zhou. When agents overtrust environmental evidence: An extensible agentic framework for benchmarking evidence-grounding defects in LLM agents, 2026. URL https://arxiv.org/abs/2605.08828.

[168] Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents, 2024. URL https://arxiv.org/abs/2406.13352.

[169] Yanxi Wang, Zhiling Zhang, Wenbo Zhou, Weiming Zhang, Jie Zhang, Qiannan Zhu, Yu Shi, Shuxin Zheng, and Jiyan He. GUIGuard-Bench: Toward a general evaluation for privacy-preserving GUI agents, 2026. URL https://arxiv.org/abs/2601.18842.

[170] Tamara Denning, Zakariya Dehlawi, and Tadayoshi Kohno. In situ with bystanders of augmented reality glasses: Perspectives on recording and privacy-mediating technologies. In *ACM CHI Conference on Human Factors in Computing Systems*, 2014. doi: 10.1145/2556288.2557352.

[171] Marion Koelle, Matthias Kranz, and Andreas Möller. Don't look at me that way! understanding user attitudes towards data glasses usage. In *ACM MobileHCI*, 2015. doi: 10.1145/2785830.2785842.

[172] Divyanshu Bhardwaj, Alexander Ponticello, Shreya Tomar, Adrian Dabrowski, and Katharina Krombholz. In focus, out of privacy: The wearer's perspective on the privacy dilemma of camera glasses. In *ACM CHI Conference on Human Factors in Computing Systems*, 2024. doi: 10.1145/3613904.3642242.

[173] Ahmed Alshehri, Joseph Spielman, Amiya Prasad, and Chuan Yue. Exploring the privacy concerns of bystanders in smart homes from the perspectives of both owners and bystanders. *Proceedings on Privacy Enhancing Technologies*, 2022(3):99–119, 2022. doi: 10.56553/popets-2022-0064. URL https://petsymposium.org/popets/2022/popets-2022-0064.php.

[174] Xiao Zhan, Guangzhi Sun, Jose Such, and Phil Woodland. Protecting bystander privacy via selective hearing in audio LLMs. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026. URL https://arxiv.org/abs/2512.06380.

[175] Mohammad Malekzadeh, Richard G. Clegg, Andrea Cavallaro, and Hamed Haddadi. Mobile sensor data anonymization. In *Proceedings of the International Conference on Internet of Things Design and Implementation*, 2019. doi: 10.1145/3302505.3310068. URL https://arxiv.org/abs/1810.11546.

[176] Nikita Kuzmin, Tao Zhong, Jiajun Deng, Yingke Zhu, Tristan Tsoi, Tianxiang Cao, Simon Lui, Kong Aik Lee, and Eng Siong Chng. Privacy-preserving end-to-end full-duplex speech dialogue models, 2026. URL https://arxiv.org/abs/2603.08179.

[177] Juan M. Coria, Hervé Bredin, Sahar Ghannay, and Sophie Rosset. Overlap-aware low-latency online speaker diarization based on end-to-end local segmentation, 2021. URL https://arxiv.org/abs/2109.06483.

[178] Vincent T. Lee, Tanfer Alan, Sung Kim, Ecenur Ustun, Amr Suleiman, Ajit Krisshna, Tim Balbekov, Armin Alaghi, and Richard Newcombe. Full system architecture modeling for wearable egocentric contextual AI, 2025. URL https://arxiv.org/abs/2512.16045.

[179] Stefanos Laskaridis, Kleomenis Katevas, Lorenzo Minto, and Hamed Haddadi. MELTing point: Mobile evaluation of language transformers, 2024. URL https://arxiv.org/abs/2403.12844.

[180] Weiwei Shan et al. A 510-nW wake-up keyword-spotting chip using serial-FFT-based MFCC and binarized depthwise separable CNN in 28-nm CMOS. *IEEE Journal of Solid-State Circuits*, 56(1):151–164, 2021. doi: 10.1109/JSSC.2020.3029097.

[181] Himax Technologies, Inc. HM01B0: Ultra low power image sensor, n.d. URL https://www.himax.com.tw/products/cmos-image-sensor/always-on-vision-sensors/hm01b0/. Product blog.

[182] Pranay Tummalapalli, Sahil Arayakandy, Ritam Pal, and Kautuk Kundan. LLM inference at the edge: Mobile, NPU, and GPU performance efficiency trade-offs under sustained load, 2026. URL https://arxiv.org/abs/2603.23640.

[183] Zhuo Chen, Wenlu Hu, Junjue Wang, Siyan Zhao, Brandon Amos, Guanhang Wu, Kiryong Ha, Khalid Elgazzar, Padmanabhan Pillai, Roberta Klatzky, Daniel Siewiorek, and Mahadev Satyanarayanan. An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC)*, 2017. doi: 10.1145/3132211.3134458.

[184] Mengwei Xu, Feng Qian, Mengze Zhu, Feifan Huang, Saumay Pushp, and Xuanzhe Liu. DeepWear: Adaptive Local Offloading for On-Wearable Deep Learning. *IEEE Transactions on Mobile Computing*, 19(2):314–330, 2020. doi: 10.1109/TMC.2019.2893250. URL https://arxiv.org/abs/1712.03073.

[185] Haoqin Tu, Jianwen Chen, Zijun Wang, Siwei Han, Juncheng Wu, Hardy Chen, Haonian Ji, Kaiwen Xiong, Jiaqi Liu, Peng Xia, Jieru Mei, Hongliang Fei, Jason Eshraghian, Zeyu Zheng, Yuyin Zhou, Huaxiu Yao, and Cihang Xie. VisualClaw: A real-time, personalized agent for the physical world, 2026. URL https://arxiv.org/abs/2606.16295.

[186] Yiyu Wang, Xuyang Liu, Xiyan Gui, Xinying Lin, Boxue Yang, Chenfei Liao, Tailai Chen, and Linfeng Zhang. Accelerating streaming video large language models via hierarchical token compression, 2026. URL https://arxiv.org/abs/2512.00891.

[187] Maria Santos-Villafranca, Jesus Bermudez-cameo, Alejandro Perez-Yus, Giovanni Maria Farinella, and Antonino Furnari. Ego-METAS: Egocentric online multimodal energy-efficient temporal action segmentation benchmark, 2026. URL https://arxiv.org/abs/2606.02246.

[188] Dingyu Yao, Junhao Zhou, Chenxu Yang, Chuanyu Qin, Haowen Hou, Zheming Liang, Congcong Wang, Yuhang Cao, Shenglong Ye, Shuai Xie, Shuhuan Gu, Haoyang Huang, Qingyi Si, Nan Duan, and Jiaqi Wang. JoyAI-VL-Interaction: Real-Time Vision-Language Interaction Intelligence, 2026. URL https://arxiv.org/abs/2606.14777.

[189] Jihao Liu, Zhiding Yu, Shiyi Lan, Shihao Wang, Rongyao Fang, Jan Kautz, Hongsheng Li, and Jose M. Alvare. StreamChat: Chatting with streaming video, 2024. URL https://arxiv.org/abs/2412.08646.

[190] Mingfei Gao, Mingze Xu, Larry S. Davis, Richard Socher, and Caiming Xiong. StartNet: Online detection of action start in untrimmed videos, 2019. URL https://arxiv.org/abs/1903.09868.

[191] Hyolim Kang, Jeongseok Hyun, Joungbin An, Youngjae Yu, and Seon Joo Kim. ActionSwitch: Class-agnostic detection of simultaneous actions in streaming videos, 2024. URL https://arxiv.org/abs/2407.12987.

[192] Hyungrok Jung, Daneul Kim, Seunggyun Lim, Jeany Son, and Jonghyun Choi. Online generic event boundary detection, 2025. URL https://arxiv.org/abs/2510.06855.

[193] Kousik Das and Debaditya Roy. Imprint: Online memory compression for long-horizon egocentric QA, 2026. URL https://arxiv.org/abs/2607.00696.

[194] Shehreen Azad, Vibhav Vineet, and Yogesh Singh Rawat. StreamReady: Learning what to answer and when in long streaming videos, 2026. URL https://arxiv.org/abs/2603.08620.

[195] Fatemehsadat Mireshghallah, Mohammadkazem Taram, Prakash Ramrakhyani, Ali Jalali, Dean Tullsen, and Hadi Esmaeilzadeh. Shredder: Learning Noise Distributions to Protect Inference Privacy. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 3–18, 2020. doi: 10.1145/3373376.3378522. URL https://mktrm.github.io/files/shredder_asplos20.pdf.

[196] Thinking Machines Lab. Interaction Models: A Scalable Approach to Human-AI Collaboration. *Thinking Machines Lab: Connectionism*, May 2026. doi: 10.64434/tml.20260511. URL https://thinkingmachines.ai/blog/interaction-models/.

[197] S M Asif Hossain, Ruksat Khan Shayoni, and Md Kishor Morol. CAPTURE: Disentangling preference drift from memory poisoning in personalized LLM agents, 2026. URL https://arxiv.org/abs/2609.02265.

[198] Baichuan Li, Junyi Yao, and Zihao Zheng. Remember, verify, or ask? evaluating memory decisions at reuse time, 2026. URL https://arxiv.org/abs/2608.19564.

[199] Xiaohongshu Dots Studio and Evolvent AI. VibeLifeBench: Can your life agent be proactive and persistent in a living world?, 2026. URL https://arxiv.org/abs/2608.10875.

[200] Toby Perrett, Ahmad Darkhalil, Saptarshi Sinha, Omar Emara, Sam Pollard, Kranti Parida, Kaiting Liu, Prajwal Gatti, Siddhant Bansal, Kevin Flanagan, Jacob Chalk, Zhifan Zhu, Rhodri Guerrier, Fahd Abdelazim, Bin Zhu, Davide Moltisanti, Michael Wray, Hazel Doughty, and Dima Damen. HD-EPIC: A highly-detailed egocentric video dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 23901–23913, 2025. doi: 10.1109/CVPR52734.2025.02226.

[201] Shaojie Zhang, Jiahui Yang, Jianqin Yin, Zhenbo Luo, and Jian Luan. Q-Frame: Query-aware frame selection and multi-resolution adaptation for Video-LLMs, 2025. URL https://arxiv.org/abs/2506.22139.

[202] Anxhelo Diko, Tinghuai Wang, Wassim Swaileh, Shiyan Sun, and Ioannis Patras. ReWind: Understanding long videos with instructed learnable memory, 2025. URL https://arxiv.org/abs/2411.15556.

[203] Senqiao Yang, Junyi Li, Xin Lai, Bei Yu, Hengshuang Zhao, and Jiaya Jia. VisionThink: Smart and Efficient Vision Language Model via Reinforcement Learning, 2025. URL https://arxiv.org/abs/2507.13348.

[204] HyperAI Team, Yuchen Liu, Kaiyang Han, Zhiqiang Xia, Yuhang Dong, Chen Song, Kangyu Tang, Jiaming Xu, Xiushi Feng, WenXuan Yu, Li Peng, Mingyang Wang, Kai Wang, Changpeng Yang, Yang Li, Haoyu Lu, Hao Wang, Bingna Xu, Guangyao Liu, Long Huang, Kaibin Guo, Jinyang Wu, Dan Wu, Hongzhen Wang, Peng Zhou, Shuai Nie, Shande Wang, Runyu Shi, and Ying Huang. HyperVL: An Efficient and Dynamic Multimodal Large Language Model for Edge Devices, 2025. URL https://arxiv.org/abs/2512.14052.

[205] Haoji Zhang, Yiqin Wang, Yansong Tang, Yong Liu, Jiashi Feng, Jifeng Dai, and Xiaojie Jin. Flash-VStream: Memory-based real-time understanding for long video streams, 2024. URL https://arxiv.org/abs/2406.08085.

[206] MOSS. MOSS-Live: Full-duplex real-time interaction, 2026. URL https://www.moss.cn/en/models/moss-live. Product page.

[207] Zhifei Xie, Zihang Liu, Ze An, Xiaobin Hu, Yue Liao, Ziyang Ma, Dongchao Yang, Mingbao Lin, Deheng Ye, Shuicheng Yan, and Chunyan Miao. Audio Interaction Model, 2026. URL https://arxiv.org/abs/2606.05121.

[208] Wujun Wen, Yunheng Li, Zhuben Dong, Lin Feng, Wanxiao Yang, and Shenlan Liu. Streaming video temporal action segmentation in real time, 2023. URL https://arxiv.org/abs/2209.13808.

[209] Jinrong Zhang, Wujun Wen, Shenglan Liu, Yunheng Li, Qifeng Li, and Lin Feng. End-to-end streaming video temporal action segmentation with reinforce learning, 2024. URL https://arxiv.org/abs/2309.15683.

[210] Qing Zhong, Guodong Ding, and Angela Yao. OnlineTAS: An online baseline for temporal action segmentation, 2024. URL https://arxiv.org/abs/2411.01122.

[211] Yuhan Shen and Ehsan Elhamifar. Progress-aware online action segmentation for egocentric procedural task videos. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[212] Roeland De Geest, Efstratios Gavves, Amir Ghodrati, Zhenyang Li, Cees Snoek, and Tinne Tuytelaars. Online action detection, 2016. URL https://arxiv.org/abs/1604.06506.

[213] Zheng Shou, Junting Pan, Jonathan Chan, Kazuyuki Miyazawa, Hassan Mansour, Anthony Vetro, Xavier Giro i Nieto, and Shih-Fu Chang. Online detection of action start in untrimmed, streaming videos, 2018. URL https://arxiv.org/abs/1802.06822.

[214] Mingze Xu, Yuanjun Xiong, Hao Chen, Xinyu Li, Wei Xia, Zhuowen Tu, and Stefano Soatto. Long short-term transformer for online action detection, 2021. URL https://arxiv.org/abs/2107.03377.

[215] Yue Zhao and Philipp Krähenbühl. Real-time online video detection with temporal smoothing transformers, 2022. URL https://arxiv.org/abs/2209.09236.

[216] Joungbin An, Yunsu Park, Hyolim Kang, and Seon Joo Kim. Object aware egocentric online action detection, 2024. URL https://arxiv.org/abs/2406.01079.

[217] Alessandro Sebastiano Catinello, Giovanni Maria Farinella, and Antonino Furnari. Mamba-OTR: a Mamba-based solution for online take and release detection from untrimmed egocentric video, 2025. URL https://arxiv.org/abs/2507.16342.

[218] Mike Zheng Shou, Stan Weixian Lei, Weiyao Wang, Deepti Ghadiyaram, and Matt Feiszli. Generic event boundary detection: A benchmark for event segmentation, 2021. URL https://arxiv.org/abs/2101.10511.

[219] Jeffrey M. Zacks, Nicole K. Speer, Khena M. Swallow, Todd S. Braver, and Jeremy R. Reynolds. Event perception: A mind/brain perspective. *Psychological Bulletin*, 133(2):273–293, 2007. doi: 10.1037/0033-2909.133.2.273.

[220] Huhai Zou, Tianhao Sun, Chuanjiang He, Yu Tian, Zhenyang Li, Li Jin, Nayu Liu, Jiang Zhong, and Kaiwen Wei. ES-Mem: Event segmentation-based memory for long-term dialogue agents, 2026. URL https://arxiv.org/abs/2601.07582.

[221] Fadime Sener, Dibyadip Chatterjee, Daniel Shelepov, Kun He, Dipika Singhania, Robert Wang, and Angela Yao. Assembly101: A large-scale multi-view video dataset for understanding procedural activities. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21064–21074, 2022. doi: 10.1109/CVPR52688.2022.02042.

[222] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Antonino Furnari, Evangelos Kazakos, Jian Ma, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, and Michael Wray. Rescaling egocentric vision: Collection, pipeline and challenges for EPIC-KITCHENS-100. *International Journal of Computer Vision*, 130 (1):33–55, 2022. doi: 10.1007/s11263-021-01531-2.

[223] Joya Chen, Ziyun Zeng, Yiqi Lin, Wei Li, Zejun Ma, and Mike Zheng Shou. LiveCC: Learning Video LLM with Streaming Speech Transcription at Scale, 2025. URL https://arxiv.org/abs/2504.16030.

[224] Md Mohaiminul Islam, Ngan Ho, Xitong Yang, Tushar Nagarajan, Lorenzo Torresani, and Gedas Bertasius. Video ReCap: Recursive captioning of hour-long videos, 2024. URL https://arxiv.org/abs/2402.13250.

[225] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 615–629, 2017. doi: 10.1145/3037697.3037698. URL https://ypkang.github.io/downloads/kang17neurosurgeon.pdf.

[226] Jon Saad-Falcon, Avanika Narayan, Robby Manihani, Tanvir Bhathal, Herumb Shandilya, Hakki Orhun Akengin, Gabriel Bo, Andrew Park, Matthew Hart, Caia Costello, Chuan Li, Christopher Ré, and Azalia Mirhoseini. OpenJarvis: Personal AI, on personal devices, 2026. URL https://arxiv.org/abs/2605.17172.

[227] Chung-Ming Chien, Manu Orsini, Eugene Kharitonov, Neil Zeghidour, Karen Livescu, and Alexandre Défossez. MoshiRAG: Asynchronous knowledge retrieval for full-duplex speech language models, 2026. URL https://arxiv.org/abs/2604.12928.

[228] Kangrui Wang, Pingyue Zhang, Zihan Wang, Yaning Gao, Linjie Li, Qineng Wang, Hanyang Chen, Chi Wan, Yiping Lu, Zhengyuan Yang, Lijuan Wang, Ranjay Krishna, Jiajun Wu, Li Fei-Fei, Yejin Choi, and Manling Li. VAGEN: Reinforcing world model reasoning for multi-turn VLM agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://arxiv.org/abs/2510.16907.

[229] Justin Kerr, Kush Hari, Ethan Weber, Chung Min Kim, Brent Yi, Tyler Bonnen, Ken Goldberg, and Angjoo Kanazawa. Eye, Robot: Learning to look to act with a BC-RL perception-action loop, 2025. URL https://arxiv.org/abs/2506.10968.

[230] Quan Wang, Ignacio Lopez Moreno, Mert Saglam, Kevin Wilson, Alan Chiao, Renjie Liu, Yanzhang He, Wei Li, Jason Pelecanos, Marily Nika, and Alexander Gruenstein. VoiceFilter-Lite: Streaming targeted voice separation for on-device speech recognition. In *Interspeech*, pages 2677–2681, 2020. doi: 10.21437/Interspeech.2020-1193. URL https://www.isca-archive.org/interspeech_2020/wang20z_interspeech.html.

[231] Yoonsang Kim, Yalong Yang, and Arie E. Kaufman. Memento: Towards proactive visualization of everyday memories with personal wearable AR assistant, 2026. URL https://arxiv.org/abs/2601.17622.

[232] Kaiqu Liang, Julia Kruk, Shengyi Qian, Xianjun Yang, Shengjie Bi, Yuanshun Yao, Shaoliang Nie, Mingyang Zhang, Lijuan Liu, Jaime Fernández Fisac, Shuyan Zhou, and Saghar Hosseini. Learning personalized agents from human feedback, 2026. URL https://arxiv.org/abs/2602.16173.

[233] Hanxiang Chao, Yihan Bai, Rui Sheng, Tianle Li, and Yushi Sun. STALE: Can LLM agents know when their memories are no longer valid?, 2026. URL https://arxiv.org/abs/2605.06527.

[234] Xuan Zhang, Yang Deng, Zifeng Ren, See-Kiong Ng, and Tat-Seng Chua. Ask-before-plan: Proactive language agents for real-world planning, 2024. URL https://arxiv.org/abs/2406.12639.

[235] Allen Z. Ren, Anushri Dixit, Alexandra Bodrova, Sumeet Singh, Stephen Tu, Noah Brown, Peng Xu, Leila Takayama, Fei Xia, Jake Varley, Zhenjia Xu, Dorsa Sadigh, Andy Zeng, and Anirudha Majumdar. Robots that ask for help: Uncertainty alignment for large language model planners. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 661–682, 2023. URL https://proceedings.mlr.press/v229/ren23a.html.

[236] Deepak Nathani, Cheng Zhang, Chang Huan, Jiaming Shan, Yinfei Yang, Alkesh Patel, Zhe Gan, William Yang Wang, Michael Saxon, and Xin Eric Wang. Proactive agent research environment: Simulating active users to evaluate proactive assistants, 2026. URL https://arxiv.org/abs/2604.00842.

[237] Kaustav Kundu, Ritvik Shrivastava, Maxim Arap, Nanshu Wang, Xianhui Zhu, Quintin Fettes, Gautam Tiwari, Parth Suresh, Prateek Singhi, Théo Moutakanni, Alejandro Castillejo Munoz, Allen Bolourchi, Pascale Fung, Pinar Donmez, Babak Damavandi, Anuj Kumar, and Seungwhan Moon. Plan, watch, recover: A benchmark and architectures for proactive procedural assistance, 2026. URL https://arxiv.org/abs/2606.04970.

[238] Matheus Kunzler Maldaner, Adam Fourney, Amanda Swearngin, Hussein Mozannar, Gagan Bansal, Maya Murad, Rafah Hosn, and Saleema Amershi. SentinelBench: A benchmark for long-running monitoring agents, 2026. URL https://arxiv.org/abs/2606.05342.

[239] Yunxin Li, Jinchao Li, Shibo Su, Zhenran Xu, Chenrui Zhao, Tongshu Bian, Xiaoman Liang, Meishan Zhang, Baotian Hu, and Min Zhang. KnowAct-GUIClaw: Know Deeply, Act Perfectly, Personal GUI Assistant with Self-Evolving Memory and Skill, 2026. URL https://arxiv.org/abs/2607.12625.

[240] Yinjie Wang, Xuyang Chen, Xiaolong Jin, Mengdi Wang, and Ling Yang. OpenClaw-RL: Train Any Agent Simply by Talking, 2026. URL https://arxiv.org/abs/2603.10165.

[241] Maithili Patel and Sonia Chernova. Robots that take initiative: A framework for building and evaluating proactive robots, 2026. URL https://arxiv.org/abs/2609.28910.

[242] SpaceXAI. Grok Bot for engineering, 2026. URL https://x.ai/bot/guides/grok-bot-for-engineering. Product blog.

[243] Jiapeng Li. Where does exactly-once live? model, harness, and tool-contract effects on duplicate side effects in LLM agents, 2026. URL https://arxiv.org/abs/2609.29095.

[244] Jeannette Bohg, Karol Hausman, Bharath Sankaran, Oliver Brock, Danica Kragic, Stefan Schaal, and Gaurav Sukhatme. Interactive perception: Leveraging action in perception and perception in action, 2017. URL https://arxiv.org/abs/1604.03670.

[245] Ri-Zhao Qiu, Bowen Wen, Yifeng Jiang, Chen Tessler, Eric Heiden, Xue Bin Peng, Xiaolong Wang, Stan Birchfield, and Erwin Coumans. ActiveAlign: Aligning egocentric manipulation via active perception, 2026. URL https://arxiv.org/abs/2610.36859.

[246] Bufang Yang, Yunqi Guo, Lilin Xu, Zhenyu Yan, Hongkai Chen, Guoliang Xing, and Xiaofan Jiang. Socialmind: Llm-based proactive ar social assistive system with human-like perception for in-situ live interactions. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 9(1):1–30, 2025.

[247] Runze Cai, Nuwan Janaka, Hyeongcheol Kim, Yang Chen, Shengdong Zhao, Yun Huang, and David Hsu. Aiget: Transforming everyday moments into hidden knowledge discovery with ai assistance on smart glasses. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–26, 2025.

[248] Adnan Abbas, Caleb Wohn, Arnav Jagtap, Eugenia H Rho, Young-Ho Kim, and Sang Won Lee. "Having lunch now": Understanding how users engage with a proactive agent for daily planning and self-reflection, 2026. URL https://arxiv.org/abs/2509.24073.

[249] Bingwen Zhu, Yuqian Fu, Qiaole Dong, Guolei Sun, Tianwen Qian, Yuzheng Wu, Danda Pani Paudel, Yanwei Fu, and Xiangyang Xue. EgoSound: Benchmarking sound understanding in egocentric videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026. URL https://arxiv.org/abs/2602.14122.

[250] Deheng Zhang, Yuqian Fu, Runyi Yang, Yang Miao, Tianwen Qian, Xu Zheng, Guolei Sun, Ajad Chhatkuli, Xuanjing Huang, Yu-Gang Jiang, Luc Van Gool, and Danda Pani Paudel. EgoNight: Towards egocentric vision understanding at night with a challenging benchmark. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2510.06218.

[251] Yanjun Li, Yuqian Fu, Tianwen Qian, Qi'Ao Xu, Silong Dai, Danda Pani Paudel, Luc Van Gool, and Xiaoling Wang. EgoCross: Benchmarking multimodal large language models for cross-domain egocentric video question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 6592–6600, 2026. doi: 10.1609/aaai.v40i8.37589. URL https://doi.org/10.1609/aaai.v40i8.37589.

[252] Qineng Wang, Wenlong Huang, Yu Zhou, Hang Yin, Tianwei Bao, Jianwen Lyu, Weiyu Liu, Ruohan Zhang, Jiajun Wu, Li Fei-Fei, and Manling Li. ENACT: Evaluating embodied cognition with world modeling of egocentric interaction, 2025. URL https://arxiv.org/abs/2511.20937.

[253] Eun Chang, Zhuangqun Huang, Yiwei Liao, Sagar Ravi Bhavsar, Amogh Param, Tammy Stark, Adel Ahmadyan, Xiao Yang, Jiaqi Wang, Ahsan Abdullah, Giang Nguyen, Akil Iyer, David Hall, Elissa Li, Shane Moon, Nicolas Scheffer, Kirmani Ahmed, Babak Damavandi, Rakesh Wanga, Anuj Kumar, Rohit Patel, and Xin Luna Dong. WearVQA: A visual question answering benchmark for wearables in egocentric authentic real-world scenarios, 2025. URL https://arxiv.org/abs/2511.22154.

[254] Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. ScreenSpot-Pro: GUI grounding for professional high-resolution computer use, 2025. URL https://arxiv.org/abs/2504.07981.

[255] Ziyao Shangguan, Chuhan Li, Yuxuan Ding, Yanan Zheng, Yilun Zhao, Tesca Fitzgerald, and Arman Cohan. TOMATO: Assessing visual temporal reasoning capabilities in multimodal foundation models, 2025. URL https://arxiv.org/abs/2410.23266.

[256] Yifei Li, Junbo Niu, Ziyang Miao, Chunjiang Ge, Yuanhang Zhou, Qihao He, Xiaoyi Dong, Haodong Duan, Shuangrui Ding, Rui Qian, Pan Zhang, Yuhang Zang, Yuhang Cao, Conghui He, and Jiaqi Wang. OVO-Bench: How far is your Video-LLM from real-world online video understanding? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025. URL https://arxiv.org/abs/2501.05510.

[257] Yujiao Shen, Shulin Tian, Jingkang Yang, and Ziwei Liu. A Simple Baseline for Streaming Video Understanding, 2026. URL https://arxiv.org/abs/2604.02317.

[258] Yulin Zhang, Cheng Shi, Yang Wang, and Sibei Yang. Eyes wide open: Ego proactive Video-LLM for streaming video, 2025. URL https://arxiv.org/abs/2510.14560.

[259] Jiaqi Yan, Ruilong Ren, Jingren Liu, Shuning Xu, Ling Wang, Yiheng Wang, Xinlin Zhong, Yun Wang, Long Zhang, Xiangyu Chen, Changzhi Sun, Jixiang Luo, Dell Zhang, Hao Sun, Chi Zhang, and Xuelong Li. TeleEgo: Benchmarking egocentric AI assistants in the wild, 2025. URL https://arxiv.org/abs/2510.23981.

[260] Dongchuan Ran, Linyu Ou, Xueheng Li, Wenwen Tong, Chenxu Guo, Hewei Guo, Kaibing Wang, and Lewei Lu. EgoPro-Bench: Benchmarking personalized proactive interaction in egocentric video streams, 2026. URL https://arxiv.org/abs/2605.07299.

[261] Yuxuan Wang, Yueqian Wang, Bo Chen, Tong Wu, Dongyan Zhao, and Zilong Zheng. Omnimmi: A comprehensive multi-modal interaction benchmark in streaming video contexts, 2025. URL https://arxiv.org/abs/2503.22952.

[262] Zhaojiang Lin, Yong Xu, Kai Sun, Jing Zheng, Yin Huang, Surya Teja Appini, Krish Narang, Renjie Tao, Ishan Kapil Jain, Siddhant Arora, Ruizhi Li, Yiteng Huang, Kaushik Patnaik, Wenfang Xu, Suwon Shon, Yue Liu, Ahmed A Aly, Anuj Kumar, Florian Metze, and Xin Luna Dong. WearVox: An egocentric multichannel voice assistant benchmark for wearables, 2025. URL https://arxiv.org/abs/2601.02391.

[263] Giacomo Spigler. TAVIS: A benchmark for egocentric active vision and anticipatory gaze in imitation learning, 2026. URL https://arxiv.org/abs/2605.07943.

[264] Yonatan Vaizman, Katherine Ellis, and Gert Lanckriet. Recognizing detailed human context in-the-wild from smartphones and smartwatches. *IEEE Pervasive Computing*, 16(4):62–74, 2017. doi: 10.1109/MPRV.2017.3971131. URL https://extrasensory.ucsd.edu/.

[265] Kristen Grauman, Andrew Westbury, Eugene Byrne, Zachary Chavis, Antonino Furnari, Rohit Girdhar, Jackson Hamburger, Hao Jiang, Miao Liu, Xingyu Liu, Miguel Martin, Tushar Nagarajan, Ilija Radosavovic, Santhosh Kumar Ramakrishnan, Fiona Ryan, Jayant Sharma, Michael Wray, Mengmeng Xu, Eric Zhongcong Xu, Chen Zhao, Siddhant Bansal, Dhruv Batra, Vincent Cartillier, Sean Crane, Tien Do, Morrie Doulaty, Akshay Erapalli, Christoph Feichtenhofer, Adriano Fragomeni, Qichen Fu, Abrham Gebreselasie, Cristina González, James Hillis, Xuhua Huang, Yifei Huang, Wenqi Jia, Weslie Khoo, Jáchym Kolář, Satwik Kottur, Anurag Kumar, Federico Landini, Chao Li, Yanghao Li, Zhenqiang Li, Karttikeya Mangalam, Raghava Modhugu, Jonathan Munro, Tullie Murrell, Takumi Nishiyasu, Will Price, Paola Ruiz, Merey Ramazanova, Leda Sari, Kiran Somasundaram, Audrey Southerland, Yusuke Sugano, Ruijie Tao, Minh Vo, Yuchen Wang, Xindi Wu, Takuma Yagi, Ziwei Zhao, Yunyi Zhu, Pablo Arbeláez, David Crandall, Dima Damen, Giovanni Maria Farinella, Christian Fuegen, Bernard Ghanem, Vamsi Krishna Ithapu, C. V. Jawahar, Hanbyul Joo, Kris Kitani, Haizhou Li, Richard Newcombe, Aude Oliva, Hyun Soo Park, James M. Rehg, Yoichi Sato, Jianbo Shi, Mike Zheng Shou, Antonio Torralba, Lorenzo Torresani, Mingfei Yan, and Jitendra Malik. Ego4D: Around the world in 3,000 hours of egocentric video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022. URL https://arxiv.org/abs/2110.07058.

[266] Yijia Lei, Jinzhao Li, Yichi Zhang, Jiacheng Hua, Yin Li, and Miao Liu. EgoSAT: A comprehensive benchmark of egocentric streaming interaction understanding, 2026. URL https://arxiv.org/abs/2606.24422.

[267] Nathan Zhao. WebPII: Benchmarking visual PII detection for computer-use agents, 2026. URL https://arxiv.org/abs/2603.17357.

[268] Xinyi Zhao, Congjing Zhang, Pei Guo, Wei Li, Lin Chen, Chaoyue Zhao, and Shuai Huang. SmartHome-Bench: A comprehensive benchmark for video anomaly detection in smart homes using multi-modal large language models, 2025. URL https://arxiv.org/abs/2506.12992. CVPR 2025 Workshop VAND 3.0.

[269] Benjamin Reichman, Xiaofan Yu, Lanxiang Hu, Jack Truxal, Atishay Jain, Rushil Chandrupatla, Tajana Šimunić Rosing, and Larry Heck. SensorQA: A question answering benchmark for daily-life monitoring, 2025. URL https://arxiv.org/abs/2501.04974.

[270] Kehan Lan, Kaining Ying, Henghui Ding, and Kaipeng Zhang. SENRI: Understanding the space-event nexus in real-life videos, 2026. URL https://arxiv.org/abs/2610.25309.

[271] Xi Chen, Julien Cumin, Fano Ramparany, and Dominique Vaufreydaz. MuRAL: A multi-resident ambient sensor dataset annotated with natural language for activities of daily living, 2025. URL https://arxiv.org/abs/2504.20505.

[272] Guan-Ting Lin, Jiachen Lian, Tingle Li, Qirui Wang, Gopala Anumanchipalli, Alexander H. Liu, and Hung yi Lee. Full-duplex-bench: A benchmark to evaluate full-duplex spoken dialogue models on turn-taking capabilities, 2025. URL https://arxiv.org/abs/2503.04721.

[273] Yueqian Wang, Xiaojun Meng, Yifan Wang, Huishuai Zhang, and Dongyan Zhao. ProactiveVideoQA: A comprehensive benchmark evaluating proactive interactions in video large language models, 2025. URL https://arxiv.org/abs/2507.09313.

[274] Jiayue Pu, Zhongxiang Sun, Zilu Zhang, Xiao Zhang, and Jun Xu. HomeSafe-Bench: Evaluating vision-language models on unsafe action detection for embodied agents in household scenarios, 2026. URL https://arxiv.org/abs/2603.11975.

[275] Yuhang Jiang. PInVerify: An offline embodied benchmark for active instance verification, 2026. URL https://arxiv.org/abs/2605.30639.

[276] Kaixuan Jiang, Yang Liu, Weixing Chen, Jingzhou Luo, Ziliang Chen, Ling Pan, Guanbin Li, and Liang Lin. Beyond the destination: A novel benchmark for exploration-aware embodied question answering, 2025. URL https://arxiv.org/abs/2503.11117.

[277] Muhammad Fadhil Ginting, Dong-Ki Kim, Xiangyun Meng, Andrzej Reinke, Bandi Jai Krishna, Navid Kayhani, Oriana Peltzer, David D. Fan, Amirreza Shaban, Sung-Kyun Kim, Mykel J. Kochenderfer, Ali-akbar Aghamohammadi, and Shayegan Omidshafiei. Enter the mind palace: Reasoning and planning for long-term active embodied question answering, 2025. URL https://arxiv.org/abs/2507.12846.

[278] Zhongbo Zhang, Jiayi Jin, Yifan Wang, Zaibin Zhang, Haiwen Diao, Lijun Wang, and Huchuan Lu. VABench: Measuring embodied spatial intelligence through visual demonstrations, active perception, and metric control, 2026. URL https://arxiv.org/abs/2609.19554.

[279] Liyang Li, Muzhi Zhu, Zhiyue Zhao, Hengyu Zhao, Ke Liu, Linhao Zhong, Hao Chen, and Chunhua Shen. Where to look: Can foundation models reach a target viewpoint through active exploration?, 2026. URL https://arxiv.org/abs/2606.01247.

[280] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 2024. URL https://arxiv.org/abs/2307.03172.

[281] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models?, 2024. URL https://arxiv.org/abs/2404.06654.

[282] Zhifei Xie, Jiaqi Lang, Ze An, Yifan Zhao, Dongchao Yang, Kai Li, Ziyang Ma, Mingbao Lin, Chunyan Miao, and Shuicheng Yan. VoiceMem: Streaming Dual-Brain Memory for Real-Time Interaction, 2026. URL https://arxiv.org/abs/2608.26005.

[283] Sitong Gong, Tianyu Yan, Caixin Kang, Bo Zheng, Xiang Ruan, Huchuan Lu, Kaipeng Zhang, Yoichi Sato, and Yifei Huang. Vinci2: Providing proactive assistance in continuous egocentric videos, 2026. URL https://arxiv.org/abs/2607.11523.

[284] Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory OS of AI agent, 2025. URL https://arxiv.org/abs/2506.06326.

[285] Shu Wang, Edwin Yu, Oscar Love, Tom Zhang, Tom Wong, Steve Scargall, and Charles Fan. MemMachine: A ground-truth-preserving memory system for personalized AI agents, 2026. URL https://arxiv.org/abs/2604.04853.

[286] Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Huajun Chen, and Ningyu Zhang. LightMem: Lightweight and Efficient Memory-Augmented Generation, 2026. URL https://arxiv.org/abs/2510.18866.

[287] Kevin Lin, Charlie Snell, Yu Wang, Charles Packer, Sarah Wooders, Ion Stoica, and Joseph E. Gonzalez. Sleep-time compute: Beyond inference scaling at test-time, 2025. URL https://arxiv.org/abs/2504.13171.

[288] Yubo Li, Yidi Miao, Yuntian Shen, and Yuxin Liu. PANDO: Efficient multimodal AI agents via online skill distillation, 2026. URL https://arxiv.org/abs/2605.24785.

[289] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory, 2024. URL https://arxiv.org/abs/2409.07429.

[290] Jiayi Tian, Shiao Liu, Yuting Xu, Jia Lu, Zihao Guan, Honglin Han, Di Yang, Minqi Gu, Yifei Qian, Tianlin Zhang, Yanqing Zhu, Zeqian Ye, Menglin Yang, Fei Wang, Xu Hu, Xiuxian Li, Wei Zhang, Shihui Su, Yiyan Ji, Jingbo Wang, Ziteng Feng, Jiaheng Liu, Zhaoxiang Zhang, Xiaolong Wu, Zixiao Tang, Zhining Gu, Yang Cai, Linbo Zheng, Jingjing Ma, Mingyang Yin, Zedong Chu, Wenbin Tang, and Mu Xu. ABot-AgentOS: A General Robotic Agent OS with Lifelong Multi-modal Memory, 2026. URL https://arxiv.org/abs/2607.10350.

[291] Leonard Bärmann, Chad DeChant, Joana Plewnia, Fabian Peller-Konrad, Daniel Bauer, Tamim Asfour, and Alex Waibel. Episodic memory verbalization using hierarchical representations of life-long robot experience, 2025. URL https://arxiv.org/abs/2409.17702.

[292] Leonard Bärmann, Joana Plewnia, Alex Waibel, and Tamim Asfour. Learning to forget: Hierarchical episodic memory for lifelong robot deployment, 2026. URL https://arxiv.org/abs/2604.11306.

[293] Haoyang Li, Yang You, Hao Su, and Leonidas Guibas. PhysMem: Scaling test-time memory for embodied physical reasoning, 2026. URL https://arxiv.org/abs/2602.20323.

[294] Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory, 2025. URL https://arxiv.org/abs/2501.13956.

[295] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory, 2025. URL https://arxiv.org/abs/2504.19413.

[296] Zicheng Zhao, Xinyang Guo, Luyao Lv, Menghan Wang, Ming Li, and Shuaicheng Li. Accurate and efficient long-term memory for LLM agents, 2026. URL https://arxiv.org/abs/2607.16211.

[297] Wei Yang, Bryce Kan, Shixuan Li, Li Li, Yuehan Qin, Jiate Li, Paul Bogdan, and Jesse Thomason. RaMem: Contextual reinstatement for long-term agentic memory, 2026. URL https://arxiv.org/abs/2606.22844.

[298] OpenAI. ChatGPT overview, 2026. URL https://openai.com/chatgpt/overview/. Product blog.

[299] Xingyu Chen, Rui Wang, Zhaopeng Tu, and Liefeng Bo. AdaMem: Learning what to remember with adaptive memory policies for personalized agents, 2026. URL https://arxiv.org/abs/2606.21144.

[300] Steven Au, Cameron J. Dimacali, Ojasmitha Pedirappagari, Namyong Park, Franck Dernoncourt, Yu Wang, Nikos Kanakaris, Hanieh Deilamsalehy, Ryan A. Rossi, and Nesreen K. Ahmed. Personalized Graph-Based Retrieval for Large Language Models, 2025. URL https://arxiv.org/abs/2501.02157.

[301] Chang Nie, Chaoyou Fu, Yifan Zhang, Haihua Yang, and Caifeng Shan. PersonaVLM: Long-term personalized multimodal LLMs, 2026. URL https://arxiv.org/abs/2604.13074.

[302] Bojie Li. User as code: Executable memory for personalized agents, 2026. URL https://arxiv.org/abs/2606.16707.

[303] Haoran Li, Jiebi Deng, Tong Jin, Jinghong Han, Yuxin Wang, Zexin Wang, Qingyi Si, Weikang Gong, Xiahai Zhuang, Jia You, Wei Cheng, Jianfeng Feng, and Hongcheng Guo. A self-evolving agent for longitudinal personal health management, 2026. URL https://arxiv.org/abs/2607.13940.

[304] María J. Pinto-Bernal, Kato Boels, and Tony Belpaeme. Fostering emotional engagement and sustained interaction in care homes through empathetic human–robot conversation. *ACM Transactions on Human-Robot Interaction*, 15(5):99, 2026. doi: 10.1145/3840294. URL https://doi.org/10.1145/3840294.

[305] Yiqun Yao, Naitong Yu, Xiang Li, Xin Jiang, Xuezhi Fang, Wenjia Ma, Xuying Meng, Jing Li, Aixin Sun, and Yequan Wang. EgoMem: Lifelong memory agent for full-duplex omnimodal models, 2026. URL https://arxiv.org/abs/2509.11914.

[306] Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas Funkhouser. TidyBot: Personalized robot assistance with large language models. *Autonomous Robots*, 47(8):1087–1102, 2023. doi: 10.1007/s10514-023-10139-z. URL https://arxiv.org/abs/2305.05658.

[307] Ali Ayub, Chrystopher L. Nehaniv, and Kerstin Dautenhahn. Interactive continual learning architecture for long-term personalization of home service robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024. URL https://arxiv.org/abs/2403.03462.

[308] Peiqi Liu, Zhanqiu Guo, Mohit Warke, Soumith Chintala, Chris Paxton, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. DynaMem: Online dynamic spatio-semantic memory for open world mobile manipulation, 2024. URL https://arxiv.org/abs/2411.04999.

[309] Zhijie Yan, Shufei Li, Zuoxu Wang, Lixiu Wu, Han Wang, Jun Zhu, Lijiang Chen, and Jihong Liu. Dynamic open-vocabulary 3D scene graphs for long-term language-guided mobile manipulation. *IEEE Robotics and Automation Letters*, 2025. URL https://arxiv.org/abs/2410.11989.

[310] Maria de Lourdes Zollo. Building Bee at Amazon: How the wearable AI device is evolving, January 2026. URL https://www.aboutamazon.com/news/devices/bee-amazon-wearable-ai-device-new-features. Product blog.

[311] Eunkyung Jo, Yuin Jeong, SoHyun Park, Daniel A. Epstein, and Young-Ho Kim. Understanding the impact of long-term memory on self-disclosure with large language model-driven chatbots for public health intervention. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, 2024. URL https://arxiv.org/abs/2402.11353.

[312] Pally Technologies, Inc. Pally, 2026. URL https://pally.com/. Product blog.

[313] Zero Email, Inc. Orchid: An AI assistant in iMessage, 2026. URL https://orchid.ai/blog/orchid-ai-assistant-imessage. Product blog.

[314] Inbox Zero. Inbox Zero, 2026. URL https://www.getinboxzero.com/. Product blog.

[315] Qinglong Shi, Donghai Wang, Hantao Zhou, Jiguo Li, Jun Xu, Jiuchong Gao, Jinghua Hao, and Renqing He. Long-term task-oriented agent: Proactive long-term intent maintenance in dynamic environments, 2026. URL https://arxiv.org/abs/2601.09382.

[316] Jinqing Zhao and Chengcan Wu. Making prospective memory SLM-Shaped: Typed intention stores for small-model agents, 2026. URL https://arxiv.org/abs/2609.01272.

[317] Qiang Zhang, Jason Naradowsky, and Yusuke Miyao. Mind the gap between conversations for improved long-term dialogue generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 10735–10762, 2023. URL https://aclanthology.org/2023.findings-emnlp.720/.

[318] Michael F. Xu, Enhui Zhao, Yawen Zhang, Joseph E. Michaelis, Sarah Sebo, and Bilge Mutlu. Designing robots for families: In-situ prototyping for contextual reminders on family routines, 2026. URL https://arxiv.org/abs/2602.22628.

[319] E. Broadbent, K. Loveys, G. Ilan, G. Chen, M. M. Chilukuri, S. G. Boardman, P. M. Doraiswamy, and D. Skuler. ElliQ, an AI-driven social robot to alleviate loneliness: progress and lessons learned. *The Journal of Aging Research and Lifestyle*, 13:22–28, 2024. doi: 10.14283/jarlife.2024.2. URL https://doi.org/10.14283/jarlife.2024.2.

[320] Toshiaki Takahashi, Shiho Higashimura, Tsuneki Ninomiya, Sachiyo Fujimura, Naoki Sugimoto, Tetsuhiko Horai, Tomoko Abe, Aya Kitamura, Masaru Matsumoto, Gojiro Nakagami, and Hiromi Sanada. The long-term use of communication robots by users of visiting healthcare services: Development of an integrated vital signs measurement system. *Cureus*, 17(1):e77635, 2025. doi: 10.7759/cureus.77635.

[321] Zhen Tao, Jinxiang Zhao, Peng Liu, Dinghao Xi, Yanfang Chen, Wei Xu, and Zhiyu Li. MemConflict: Evaluating long-term memory systems under memory conflicts, 2026. URL https://arxiv.org/abs/2605.20926.

[322] Ona. Ona: AI wearable note taker and voice recorder, 2026. URL https://onavoice.ai/. Product page.

[323] Forth. Forth Band and Cue, 2026. URL https://getforth.co/facts/. Product page.

[324] Genglin Liu and Saadia Gabriel. PM-Bench: Evaluating prospective memory in LLM agents, 2026. URL https://arxiv.org/abs/2607.12385.

[325] Tianhua Zhang, Xinjiang Wang, Qianxi Zhang, Qi Chen, Kun Li, Yaoqi Chen, Dingdong Wang, Helen Meng, and Yan Lu. TriggerBench: Investigating prospective memory for large language models, 2026. URL https://arxiv.org/abs/2606.23459.

[326] B. Y. Yan, Chaofan Li, Hongjin Qian, Shuqi Lu, and Zheng Liu. General Agentic Memory Via Deep Research, 2025. URL https://arxiv.org/abs/2511.18423.

[327] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023. URL https://arxiv.org/abs/2305.16291.

[328] Zeyu Zhang, Ziliang Guo, Yihang Sun, Xichong Zhang, Xixuan Hao, Zehao Lin, Yang Zhang, Xiaoyan Zhao, Tong Shen, Bo Tang, Zhi-Qin John Xu, Junchi Yan, Haofen Wang, Xu Chen, Feiyu Xiong, Zhiyu Li, and Tat-Seng Chua. Metis: Memory Foundation Model, 2026. URL https://arxiv.org/abs/2607.26760.

[329] Yu Wang, Yifan Gao, Xiusi Chen, Haoming Jiang, Shiyang Li, Jingfeng Yang, Qingyu Yin, Zheng Li, Xian Li, Bing Yin, Jingbo Shang, and Julian McAuley. MEMORYLLM: Towards self-updatable large language models, 2024. URL https://arxiv.org/abs/2402.04624.

[330] Zhiyu Li, Chenyang Xi, Chunyu Li, Ding Chen, Boyu Chen, Shichao Song, Simin Niu, Hanyu Wang, Jiawei Yang, Chen Tang, Qingchen Yu, Jihao Zhao, Yezhaohui Wang, Peng Liu, Zehao Lin, Pengyuan Wang, Jiahao Huo, Tianyi Chen, Kai Chen, Kehang Li, Zhen Tao, Huayi Lai, Hao Wu, Bo Tang, Zhengren Wang, Zhaoxin Fan, Ningyu Zhang, Linfeng Zhang, Junchi Yan, Mingchuan Yang, Tong Xu, Wei Xu, Huajun Chen, Haofen Wang, Hongkang Yang, Wentao Zhang, Zhi-Qin John Xu, Siheng Chen, and Feiyu Xiong. MemOS: A memory OS for AI system, 2025. URL https://arxiv.org/abs/2507.03724.

[331] Wei-Chieh Huang, Weizhi Zhang, Yuchen Wu, Yankai Chen, Eric Hanchen Jiang, Wooseong Yang, Yiwei Yang, Henry Peng Zou, Hanrong Zhang, Ying Nian Wu, Haolun Wu, Kai-Wei Chang, Philip S. Yu, Xue Liu, and Aylin Caliskan. Harness the memory: A holistic evaluation of memory substrates in memory agents, 2026. URL https://arxiv.org/abs/2608.15008.

[332] OpenClaw. OpenClaw: Open-source AI assistant, 2026. URL https://openclaw.ai/. Product blog.

[333] Nous Research. Hermes Agent: Open-source AI agent that grows with you, 2026. URL https://hermes-agent.nousresearch.com/. Product blog.

[334] Yining Chen, Jihao Zhao, Bo Tang, Haofen Wang, Yue Zhang, Fei Huang, Feiyu Xiong, and Zhiyu Li. MemPrivacy: Privacy-Preserving Personalized Memory Management for Edge-Cloud Agents, 2026. URL https://arxiv.org/abs/2605.09530.

[335] Guanzhen Li, Liangming Pan, and Leye Wang. ProEvent: An event-centric benchmark for proactive agents, 2026. URL https://arxiv.org/abs/2607.17701.

[336] Tianxiang Fei, Mingyang Song, Mao Zheng, and Xiang Yu. Memory beyond recall: A dual-process cognitive memory system for self-evolving LLM agents, 2026. URL https://arxiv.org/abs/2606.09483.

[337] Md Nayem Uddin, Kumar Shubham, Eduardo Blanco, Chitta Baral, and Gengyu Wang. From recall to forgetting: Benchmarking long-term memory for personalized agents, 2026. URL https://arxiv.org/abs/2604.20006.

[338] Yeonjun In, Wonjoong Kim, Sangwu Park, Kanghoon Yoon, and Chanyoung Park. Personalize-then-store: Benchmarking and learning personalized memory for long-horizon agents, 2026. URL https://arxiv.org/abs/2605.25535.

[339] Xiaoyu Xu, Minxin Du, Qipeng Xie, Haobin Ke, Qingqing Ye, and Haibo Hu. When routine chats turn toxic: Unintended long-term state poisoning in personalized agents, 2026. URL https://arxiv.org/abs/2605.06731.

[340] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023. URL https://arxiv.org/abs/2304.03442.

[341] Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. MEM1: Learning to synergize memory and reasoning for efficient long-horizon agents, 2025. URL https://arxiv.org/abs/2506.15841.

[342] Suozhao Ji, Baodong Wu, Zehao Wang, Lei Xia, Qingping Li, Ruisong Wang, Wenbo Ding, Zhenhua Zhu, Boxun Li, Guohao Dai, and Yu Wang. Infini memory: Maintainable topic documents for long-term LLM agent memory, 2026. URL https://arxiv.org/abs/2606.10677.

[343] Chris Latimer, Nicoló Boschi, Andrew Neeser, Chris Bartholomew, Gaurav Srivastava, Xuan Wang, and Naren Ramakrishnan. Hindsight is 20/20: Building agent memory that retains, recalls, and reflects, 2025. URL https://arxiv.org/abs/2512.12818.

[344] Weixiang Zhao, Yingshuo Wang, Yichen Zhang, Yanyan Zhao, Yu Zhang, Yang Wu, Dandan Tu, Bing Qin, and Ting Liu. Rethinking experience utilization in self-evolving language model agents, 2026. URL https://arxiv.org/abs/2605.07164.

[345] Wenya Xie, Shengming Zhou, Zelin Li, Pouya Parsa, Shuang Zhou, Xinheng Ding, Chinmay Arvind, Guanchu Wang, Vladimir Braverman, Ali Payani, Yantao Zheng, and Zirui Liu. DynamicMem: A long-horizon memory benchmark in real-world settings, 2026. URL https://arxiv.org/abs/2606.22877.

[346] Ce Zhang, Taixi Lu, Md Mohaiminul Islam, Ziyang Wang, Shoubin Yu, Mohit Bansal, and Gedas Bertasius. A simple LLM framework for long-range video question-answering, 2024. URL https://arxiv.org/abs/2312.17235.

[347] Ziyang Wang, Shoubin Yu, Elias Stengel-Eskin, Jaehong Yoon, Feng Cheng, Gedas Bertasius, and Mohit Bansal. VideoTree: Adaptive tree-based video representation for LLM reasoning on long videos, 2025. URL https://arxiv.org/abs/2405.19209.

[348] Muzhao Tian, Zisu Huang, Xiaohua Wang, Jingwen Xu, Zhengkang Guo, Qi Qian, Yuanzhe Shen, Kaitao Song, Jiakang Yuan, Changze Lv, and Xiaoqing Zheng. Controllable memory usage: Balancing anchoring and innovation in long-term human-agent interaction, 2026. URL https://arxiv.org/abs/2601.05107.

[349] Yulin Hu, Zimo Long, Jiahe Guo, Xingyu Sui, Xing Fu, Weixiang Zhao, Yanyan Zhao, and Bing Qin. OP-Bench: Benchmarking over-personalization for memory-augmented personalized conversational agents, 2026. URL https://arxiv.org/abs/2601.13722.

[350] Yixiong Chen, Xinyi Bai, and Alan Yuille. The compliance trap: Diagnosing how AI agents consume conflicting memory, 2026. URL https://arxiv.org/abs/2607.10608.

[351] Caddy Technologies. Caddy: Important email alerts, 2026. URL https://caddy.app/use-cases/important-email-alerts. Product blog.

[352] Chongrui Ye, Yuxiang Liu, Yu Wang, Haofei Yu, Yining Zhao, Ge Liu, Julian McAuley, and Jiaxuan You. Auto-Dreamer: Learning offline memory consolidation for language agents, 2026. URL https://arxiv.org/abs/2605.20616.

[353] Junwei Liao, Haoting Shi, Ruiwen Zhou, Jiaqian Wang, Shengtao Zhang, Wei Zhang, Ying Wen, Zhiyu Li, Feiyu Xiong, Bo Tang, Weinan Zhang, and Muning Wen. MemQ: Integrating Q-learning into self-evolving memory agents over provenance DAGs, 2026. URL https://arxiv.org/abs/2605.08374.

[354] Suli Wang, Yiqun Duan, Yu Deng, Rundong Zhao, Dai Shi, Minghua Deng, Chen Chen, Yiqi Wang, and Xinliang Zhou. Cognifold: Always-on proactive memory via cognitive folding, 2026. URL https://arxiv.org/abs/2605.13438.

[355] Dylan Zhang, Yanshan Lin, Zhengkun Wu, Yihang Sun, Bingxuan Li, Dianqi Li, and Hao Peng. Useful memories become faulty when continuously updated by LLMs, 2026. URL https://arxiv.org/abs/2605.12978.

[356] Jianjie Zheng, Zhichen Liu, Zhanyu Shen, Jingxiang Qu, Guanhua Chen, Yile Wang, Yang Xu, Yang Liu, and Sijie Cheng. Evaluating memory capability in continuous lifelog scenario, 2026. URL https://arxiv.org/abs/2604.11182.

[357] Haofei Sun and Lin He. When memory updates but behavior does not: Repairing implicit stale dependencies in personalized agent responses, 2026. URL https://arxiv.org/abs/2608.01619.

[358] Nozomio Labs. folk: An AI that nags you, 2026. URL https://www.folk.com/use-cases/ai-that-nags-you. Product blog.

[359] Xinyi Fan, Miri Liu, Ruozhen Yang, Siru Ouyang, and Jiawei Han. Can agent memory systems track evolving state?, 2026. URL https://arxiv.org/abs/2608.19652.

[360] Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. MemSkill: Learning and evolving memory skills for self-evolving agents, 2026. URL https://arxiv.org/abs/2602.02474.

[361] Guanming Liu, Yuqi Ren, Hansu Gu, Peng Zhang, Weihang Wang, Jiahao Liu, Ning Gu, and Tun Lu. StreamMemBench: Streaming evaluation of agent memory for future-oriented assistance, 2026. URL https://arxiv.org/abs/2606.14571.

[362] Intuition Robotics. ElliQ: Companion robot for older adults, 2026. URL https://elliq.com/. Product blog.

[363] Jiayang Cheng, Dongyu Ru, Lin Qiu, Yiyang Li, Xuezhi Cao, Yangqiu Song, and Xunliang Cai. AMemGym: Interactive memory benchmarking for assistants in long-horizon conversations. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/0856bc553d3e3b9827e5140d0ad3bf8d-Abstract-Conference.html.

[364] Chengzhi Liu, Yuzhe Yang, Sophia Xiao Pu, Yepeng Liu, Lin Long, Yichen Guo, Nuo Chen, Zhaotian Weng, Elena Kochkina, Simerjot Kaur, Charese Smiley, Xiaomo Liu, James Zou, Sheng Liu, Yuheng Bu, Songyou Peng, and Xin Eric Wang. WorldMemArena: Evaluating multimodal agent memory through action-world interaction, 2026. URL https://arxiv.org/abs/2605.29341.

[365] Zhen Tan, Jun Yan, I-Hung Hsu, Rujun Han, Zifeng Wang, Long Le, Yiwen Song, Yanfei Chen, Hamid Palangi, George Lee, et al. In prospect and retrospect: Reflective memory management for long-term personalized dialogue agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8416–8439, 2025.

[366] Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schütze, Volker Tresp, and Yunpu Ma. Memory-R1: Enhancing large language model agents to manage and utilize memories via reinforcement learning, 2026. URL https://arxiv.org/abs/2508.19828.

[367] Sikuan Yan, Ahmed Bahloul, Ercong Nie, Susanna Schwarzmann, Riccardo Trivisonno, Volker Tresp, and Yunpu Ma. Memory-R2: Fair credit assignment for long-horizon memory-augmented LLM agents, 2026. URL https://arxiv.org/abs/2605.21768.

[368] Prince Zizhuang Wang and Shuli Jiang. PRIME: Training free proactive reasoning via iterative memory evolution for user-centric agent, 2026. URL https://arxiv.org/abs/2604.07645.

[369] Jiho Kim, Junseong Choi, Woosog Chay, Daeun Kyung, Yeonsu Kwon, Yohan Jo, and Edward Choi. ProPerSim: Developing proactive and personalized AI assistants through user-assistant simulation, 2026. URL https://arxiv.org/abs/2509.21730.

[370] Adiba Orzikulova, Han Xiao, Zhipeng Li, Yukang Yan, Yuntao Wang, Yuanchun Shi, Marzyeh Ghassemi, Sung-Ju Lee, Anind K. Dey, and Xuhai Xu. Time2Stop: Adaptive and explainable human-AI loop for smartphone overuse intervention. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 2024. doi: 10.1145/3613904.3642747. URL https://arxiv.org/abs/2403.05584.

[371] Yi Yang, Zhennan Chen, Yihong Zhuang, Tiehan Fan, Yinan Chen, Jian Li, Jian Yang, and Ying Tai. RoMeRL: Balancing feedback coverage and the memory-reward trap in self-evolving agent memory via reduced-order utility states, 2026. URL https://arxiv.org/abs/2608.02508.

[372] Di Wu, Zixiang Ji, Asmi Kawatkar, Bryan Kwan, Jia-Chen Gu, Nanyun Peng, and Kai-Wei Chang. LongMemEval-V2: Evaluating long-term agent memory toward experienced colleagues, 2026. URL https://arxiv.org/abs/2605.12493.

[373] Wei Zhou, Xuanhe Zhou, Shaokun Han, Hongming Xu, Guoliang Li, Zhiyu Li, Feiyu Xiong, and Fan Wu. Are we ready for an agent-native memory system?, 2026. URL https://arxiv.org/abs/2606.24775.

[374] Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents, 2024. URL https://arxiv.org/abs/2402.17753.

[375] Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory, 2025. URL https://arxiv.org/abs/2410.10813.

[376] Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in LLM agents via incremental multi-turn interactions. *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2507.05257.

[377] Xiyu Ren, Zhaowei Wang, Yiming Du, Zhongwei Xie, Chi Liu, Xinlin Yang, Haoyue Feng, Wenjun Pan, Tianshi Zheng, Baixuan Xu, Zhengnan Li, Yangqiu Song, Ginny Wong, and Simon See. MemLens: Benchmarking Multimodal Long-Term Memory in Large Vision-Language Models, 2026. URL https://arxiv.org/abs/2605.14906.

[378] Minghao Guo, Qingyue Jiao, Zeru Shi, Yihao Quan, Boxuan Zhang, Danrui Li, Liwei Che, Wujiang Xu, Shilong Liu, Zirui Liu, Mubbasir Kapadia, Vladimir Pavlovic, Jiang Liu, Mengdi Wang, Yiyu Shi, Dimitris N. Metaxas, and Ruixiang Tang. MemEye: A Visual-Centric Evaluation Framework for Multimodal Agent Memory, 2026. URL https://arxiv.org/abs/2605.15128.

[379] Ziyang Wang, Yue Zhang, Shoubin Yu, Ce Zhang, Zengqi Zhao, Jaehong Yoon, Hyunji Lee, Gedas Bertasius, and Mohit Bansal. EgoMemReason: A memory-driven reasoning benchmark for long-horizon egocentric video understanding, 2026. URL https://arxiv.org/abs/2605.09874.

[380] Karmesh Yadav, Yusuf Ali, Gunshi Gupta, Yarin Gal, and Zsolt Kira. FindingDory: A benchmark to evaluate memory in embodied agents, 2025. URL https://arxiv.org/abs/2506.15635.

[381] Ding Chen, Simin Niu, Kehang Li, Peng Liu, Xiangping Zheng, Bo Tang, Xinchi Li, Feiyu Xiong, and Zhiyu Li. HaluMem: Evaluating Hallucinations in Memory Systems of Agents, 2026. URL https://arxiv.org/abs/2511.03506.

[382] Xixuan Hao, Zeyu Zhang, Zehao Lin, Yihang Sun, Ziliang Guo, Xichong Zhang, Yuxuan Liang, Feiyu Xiong, and Zhiyu Li. MemOps: Benchmarking lifecycle memory operations in long-horizon conversations, 2026. URL https://arxiv.org/abs/2607.12893.

[383] Junxiang Wang, Yuchen Liu, Yi Wu, Guofei Chen, Nir Pechuk, Tao Chen, Zackory Erickson, Yuzhe Qin, and Tianyu Li. What should embodied agents remember? benchmarking and evaluating memory modality for embodied agents, 2026. URL https://arxiv.org/abs/2610.12561.

[384] Hyunji Lee, Justin Chih-Yao Chen, Joykirat Singh, Zaid Khan, Elias Stengel-Eskin, and Mohit Bansal. MINTEval: Evaluating memory under multi-target interference in long-horizon agent systems, 2026. URL https://arxiv.org/abs/2605.18565.

[385] Yehang Zhang, Jianchong Su, Haojian Huang, Yifan Chang, Tianhao Zhou, Xinli Xu, Yingjie Xu, Yinchuan Li, Zexi Li, and Ying-Cong Chen. WorldLines: Benchmarking and modeling long-horizon stateful embodied agents, 2026. URL https://arxiv.org/abs/2606.18847.

[386] Yian Wang, Agam Goyal, Yuen Chen, and Hari Sundaram. State contamination in memory-augmented LLM agents, 2026. URL https://arxiv.org/abs/2605.16746.

[387] Xutao Mao, Liangjie Zhao, Leyao Wang, Rui Qian, Qiang Huang, Wentao Wang, Bo Han, Xiang Zheng, and Cong Wang. Agents don't just agree, they remember: Benchmarking persistent sycophancy in self-improving personal agents, 2026. URL https://arxiv.org/abs/2607.10526.

[388] Taeyoon Kwon, Dongwook Choi, Hyojun Kim, Sunghwan Kim, Seungjun Moon, Beong woo Kwak, Kuan-Hao Huang, and Jinyoung Yeo. Embodied Agents Meet Personalization: Investigating Challenges and Solutions Through the Lens of Memory Utilization, 2026. URL https://arxiv.org/abs/2505.16348.

[389] Maosheng Qin, Qi Zhang, Mingxuan Xia, Jinglei Chen, Jiexiang Wang, Sheng Guo, Cheng Peng, and Haobo Wang. Memorizing who, knowing how: Benchmarking personal and procedural memory for persistent assistants, 2026. URL https://arxiv.org/abs/2610.20022.

[390] Xidong Yang, Xingyi Zhang, Wenhao Li, Wenyan Liu, Junjie Sheng, Yun Hua, Wei Yin, Tao Fang, Chuyun Shen, and Xiangfeng Wang. PATH-Bench: Path-dependent evaluation of lifelong agents, 2026. URL https://arxiv.org/abs/2608.01149.

[391] Parth Asawa, Christopher M. Glaze, Gabriel Orlanski, Ramya Ramakrishnan, Benji Xu, Asim Biswal, Vincent Sunn Chen, Frederic Sala, Matei Zaharia, and Joseph E. Gonzalez. Continual learning bench: Evaluating frontier AI systems in real-world stateful environments, 2026. URL https://arxiv.org/abs/2606.05661.

[392] Tongbo Chen, Zhengxi Lu, Zhan Xu, Guocheng Shao, Shaohan Zhao, Fei Tang, Yong Du, Kaitao Song, Yizhou Liu, Yuchen Yan, Wenqi Zhang, Xu Tan, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. KnowU-Bench: Towards interactive, proactive, and personalized mobile agent evaluation, 2026. URL https://arxiv.org/abs/2604.08455.

[393] Zhe Yang, Shulin Tian, Kairui Hu, Shuai Liu, Hoang-Nhat Nguyen, Yichi Zhang, Zujin Guo, Mengying Yu, Zinan Zhang, Jingkang Yang, Chen Change Loy, and Ziwei Liu. HippoCamp: Benchmarking contextual agents on personal computers, 2026. URL https://arxiv.org/abs/2604.01221.

[394] Shuai Liu, Shulin Tian, Kairui Hu, Yuhao Dong, Zhe Yang, Bo Li, Jingkang Yang, Chen Change Loy, and Ziwei Liu. FileGram: Grounding agent personalization in file-system behavioral traces, 2026. URL https://arxiv.org/abs/2604.04901.

[395] Jialin Li, Zhenhao Chen, Hanjun Luo, and Hanan Salam. Why bother me again? benchmarking personal agent memory for interaction preferences, 2026. URL https://arxiv.org/abs/2610.30485.

[396] Yusong Lin, Xinyuan Liang, Haiyang Wang, Qipeng Gu, Siqi Cheng, Jiangui Chen, Shuzhe Wu, Feiyang Pan, Lue Fan, Sanyuan Zhao, and Dandan Tu. Claw-anything: Benchmarking always-on personal assistants with broader access to user's digital world, 2026. URL https://arxiv.org/abs/2605.26086.

[397] Qinglong Yang, Haoming Li, Haotian Zhao, Xiaokai Yan, Jingtao Ding, Fengli Xu, and Yong Li. Fingertip 20k: A benchmark for proactive and personalized mobile llm agents, 2026. URL https://arxiv.org/abs/2507.21071.

[398] Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, ZhongZhi Li, Yingying Zhang, Le Song, and Qianli Ma. LifelongAgentBench: Evaluating LLM agents as lifelong learners, 2025. URL https://arxiv.org/abs/2505.11942.

[399] Yuxin Chen, Yi Zhang, Zhengzhou Cai, Yaorui Shi, Zhiyuan Yao, Chenhang Cui, Jingnan Zheng, Yaqi Huo, Xi Su, Qi Gu, Xunliang Cai, Xiang Wang, An Zhang, and Tat-Seng Chua. VitaBench 2.0: Evaluating personalized and proactive agents in long-term user interactions, 2026. URL https://arxiv.org/abs/2605.27141.

[400] Yuyao Wang, Zhongjian Zhang, Mo Chi, Kaichi Yu, Yuhan Li, Miao Peng, Bing Tong, Chen Zhang, Yan Zhou, and Jia Li. EvoMemBench: Benchmarking agent memory from a self-evolving perspective, 2026. URL https://arxiv.org/abs/2605.18421.

[401] Jiahao Huang, Fei Cheng, Junfeng Jiang, Zefan Yu, and Akiko Aizawa. BenchTrace: A benchmark for testing reflection ability and controlled evolution in LLM agents, 2026. URL https://arxiv.org/abs/2605.29225.

[402] Tianyi Guan, Yiding Wang, Haotong Yang, Siyuan Cao, Shirui Liu, Yi Hu, Jiaqi Li, and Muhan Zhang. ContinualSkillBench: Can LLM agents truly evolve their capabilities?, 2026. URL https://arxiv.org/abs/2608.03874.

[403] Shengyuan Ding, Xilin Wei, Xinyu Fang, Haodong Duan, Dahua Lin, Jiaqi Wang, and Yuhang Zang. Beyond the Current Observation: Evaluating Multimodal Large Language Models in Controllable Non-Markov Games, 2026. URL https://arxiv.org/abs/2606.19338.

[404] Jenny Yang, Elizabeth Proehl, Anjan Bharadwaj, Emily Fan, Charlotte Cole, James Kim, Noam Brown, Phoebe Thacker, Amelia Glaese, and Tejal Patwardhan. LifeVal: Evaluating AI model performance on personalized agentic consumer tasks. Technical report, OpenAI, 2026. URL https://cdn.openai.com/pdf/lifeval.pdf.

[405] Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, Wanjun Zhong, Yining Ye, Yujia Qin, Yuwen Xiong, Yuxin Song, Zhiyong Wu, Aoyan Li, Bo Li, Chen Dun, Chong Liu, Daoguang Zan, Fuxing Leng, Hanbin Wang, Hao Yu, Haobin Chen, Hongyi Guo, Jing Su, Jingjia Huang, Kai Shen, Kaiyu Shi, Lin Yan, Peiyao Zhao, Pengfei Liu, Qinghao Ye, Renjie Zheng, Shulin Xin, Wayne Xin Zhao, Wen Heng, Wenhao Huang, Wenqian Wang, Xiaobo Qin, Yi Lin, Youbin Wu, Zehui Chen, Zihao Wang, Baoquan Zhong, Xinchun Zhang, Xujing Li, Yuanfan Li, Zhongkai Zhao, Chengquan Jiang, Faming Wu, Haotian Zhou, Jinlin Pang, Li Han, Qi Liu, Qianli Ma, Siyao Liu, Songhua Cai, Wenqi Fu, Xin Liu, Yaohui Wang, Zhi Zhang, Bo Zhou, Guoliang Li, Jiajun Shi, Jiale Yang, Jie Tang, Li Li, Qihua Han, Taoran Lu, Woyu Lin, Xiaokang Tong, Xinyao Li, Yichi Zhang, Yu Miao, Zhengxuan Jiang, Zili Li, Ziyuan Zhao, Chenxin Li, Dehua Ma, Feng Lin, Ge Zhang, Haihua Yang, Hangyu Guo, Hongda Zhu, Jiaheng Liu, Junda Du, Kai Cai, Kuanye Li, Lichen Yuan, Meilan Han, Minchao Wang, Shuyue Guo, Tianhao Cheng, Xiaobo Ma, Xiaojun Xiao, Xiaolong Huang, Xinjie Chen, Yidi Du, Yilin Chen, Yiwen Wang, Zhaojian Li, Zhenzhu Yang, Zhiyuan Zeng, Chaolin Jin, Chen Li, Hao Chen, Haoli Chen, Jian Chen, Qinghao Zhao, and Guang Shi. UI-TARS-2 technical report: Advancing GUI agent with multi-turn reinforcement learning, 2025. URL https://arxiv.org/abs/2509.02544.

[406] Huang Fang, Mengxi Zhang, Heng Dong, Wei Li, Zixuan Wang, Qifeng Zhang, Xueyun Tian, Yucheng Hu, and Hang Li. Robix: A Unified Model for Robot Interaction, Reasoning and Planning, 2025. URL https://arxiv.org/abs/2509.01106.

[407] Albert Yu, Chengshu Li, Luca Macesanu, Arnav Balaji, Ruchira Ray, Raymond Mooney, and Roberto Martín-Martín. Mixed-initiative dialog for human-robot collaborative manipulation. In *International Conference on Robotics and Automation (ICRA)*, June 2026. URL https://arxiv.org/abs/2508.05535.

[408] Manus. Scheduled tasks 2.0, May 2026. URL https://manus.im/blog/manus-schedules. Product blog.

[409] Brea AI LLC. Brea: The AI that gets things done, 2026. URL https://brea.ai/. Product blog.

[410] Anthropic. Claude, 2026. URL https://claude.com/product/overview. Product blog.

[411] LangChain. Introducing ambient agents, 2025. URL https://www.langchain.com/blog/introducing-ambient-agents. Executive AI Assistant and Agent Inbox reference implementation.

[412] Google. The Gemini app becomes more agentic, delivering proactive, 24/7 help, May 2026. URL https://blog.google/innovation-and-ai/products/gemini-app/next-evolution-gemini-app/. Product blog.

[413] Simone Gallo, Fabio Paternò, and Alessio Malizia. A conversational agent for creating automations exploiting large language models. *Personal and Ubiquitous Computing*, 28:931–946, 2024. doi: 10.1007/s00779-024-01825-5. URL https://doi.org/10.1007/s00779-024-01825-5.

[414] Han Wool Jung, Jooho Lee, Jin Young Park, Woo Jung Kim, and Jaesub Park. Detection of depression risk among older adults using home-deployed socially assistive robots: A real-world study. *Frontiers in Psychiatry*, 17:1740539, 2026. doi: 10.3389/fpsyt.2026.1740539.

[415] Evan King, Haoxiang Yu, Sangsu Lee, and Christine Julien. Sasha: Creative goal-oriented reasoning in smart homes with large language models. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 8:1–38, 2024. doi: 10.1145/3643505. URL https://arxiv.org/abs/2305.09802.

[416] Tuochao Chen, Nicholas Scott Batchelder, Alisa Liu, Noah A Smith, and Shyamnath Gollakota. LLAMAPIE: Proactive in-ear conversation assistants. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 13801–13824, 2025.

[417] Yueqian Wang, Songxiang Liu, Disong Wang, Nuo Xu, Guanglu Wan, Huishuai Zhang, and Dongyan Zhao. MMDuet2: Enhancing proactive interaction of video MLLMs with multi-turn reinforcement learning, 2025. URL https://arxiv.org/abs/2512.06810.

[418] Geonsun Lee, Min Xia, Nels Numan, Xun Qian, David Li, Yanhe Chen, Achin Kulshrestha, Ishan Chatterjee, Yinda Zhang, Dinesh Manocha, David Kim, and Ruofei Du. Sensible agent: A framework for unobtrusive interaction with proactive AR agents. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22, 2025. doi: 10.1145/3746059.3747748. URL https://arxiv.org/abs/2509.09255.

[419] Heeseung Kim. EgoVoice: Proactive spoken assistance from egocentric multimodal streams, 2026. URL https://arxiv.org/abs/2610.12248.

[420] Siqi Guo, Yuxia Chen, Wenbo Zhu, Yangguang Ji, Zhenxiang Jiang, Teng Zhong, and Yu Zhang. PACE: Plan, attend, check, and emit for training-free response timing in streaming video understanding, 2026. URL https://arxiv.org/abs/2610.28323.

[421] Huawei. Xiaoyi suggestions: Frequently asked questions, 2026. URL https://consumer.huawei.com/cn/support/content/zh-cn15965298/. Product blog.

[422] Lei Ding, Bin He, Chenguang Wang, and Yang Liu. ProActor: Timing-aware reinforcement learning for proactive task scheduling agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, 2026. URL https://aclanthology.org/2026.acl-long.832/.

[423] Adrian Aguilera, Marvyn Arévalo Avalos, Jing Xu, Bibhas Chakraborty, Caroline Figueroa, Faviola Garcia, Karina Rosales, Rosa Hernandez-Ramos, Chris Karr, Joseph Williams, Lisa Ochoa-Frongia, Urmimala Sarkar, Elad Yom-Tov, and Courtney Lyles. Effectiveness of a digital health intervention leveraging reinforcement learning: Results from the diabetes and mental health adaptive notification tracking and evaluation (DIAMANTE) randomized clinical trial. *Journal of Medical Internet Research*, 26:e60834, 2024. doi: 10.2196/60834.

[424] Anna L. Trella, Kelly W. Zhang, Hinal Jajal, Inbal Nahum-Shani, Vivek Shetty, Finale Doshi-Velez, and Susan A. Murphy. A deployed online reinforcement learning algorithm in an oral health clinical trial. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 28792–28800, 2025. doi: 10.1609/aaai.v39i28.35143. URL https://arxiv.org/abs/2409.02069.

[425] Susobhan Ghosh, Yongyi Guo, Pei-Yao Hung, Lara Coughlin, Erin Bonar, Inbal Nahum-Shani, Maureen Walton, and Susan Murphy. MiWaves reinforcement learning algorithm, 2024. URL https://arxiv.org/abs/2408.15076.

[426] Sven Eberhardt. The science behind Hunches: Deep device embeddings, 2022. URL https://www.amazon.science/blog/the-science-behind-hunches-deep-device-embeddings. Product blog.

[427] Siyin Wang, Jinlan Fu, Feihong Liu, Xinzhe He, Huangxuan Wu, Junhao Shi, Kexin Huang, Zhaoye Fei, Jingjing Gong, Zuxuan Wu, Yu-Gang Jiang, See-Kiong Ng, Tat-Seng Chua, and Xipeng Qiu. RoboOmni: Proactive Robot Manipulation in Omni-modal Context, 2025. URL https://arxiv.org/abs/2510.23763.

[428] Chengbo He, Sheng Li, Chenyang Ma, Bochao Zou, Li Sun, Jiansheng Chen, Junliang Xing, Yuanchun Shi, and Huimin Ma. PACT: Proactive asking for continual task assistance in human-robot collaboration, 2026. URL https://arxiv.org/abs/2605.24350.

[429] Leo Xu, Letian Li, Alex Cuellar, and Michael Hagenow. Learning to assist: Collaborative VLAs for implicit human-robot collaboration, 2026. URL https://arxiv.org/abs/2606.12475.

[430] Yueqian Wang, Xiaojun Meng, Yuxuan Wang, Jianxin Liang, Jiansheng Wei, Huishuai Zhang, and Dongyan Zhao. VideoLLM knows when to speak: Enhancing time-sensitive video comprehension with video-text duet interaction format, 2024. URL https://arxiv.org/abs/2411.17991.

[431] Kratika Bhagtani, Mrinal Anand, Yu Chen Xu, and Amit Kumar Singh Yadav. Speak or stay silent: Context-aware turn-taking in multi-party dialogue, 2026. URL https://arxiv.org/abs/2603.11409.

[432] Vihaan Nama, Shreya Mendi, Zian Ye, and Brinnae Bent. When2Speak: A dataset for temporal participation and turn-taking in multi-party conversations for large language models, 2026. URL https://arxiv.org/abs/2605.05626.

[433] Predrag Klasnja, Shawna Smith, Nicholas J. Seewald, Andy Lee, Kelly Hall, Brook Luers, Eric B. Hekler, and Susan A. Murphy. Efficacy of contextually tailored suggestions for physical activity: A micro-randomized optimization trial of HeartSteps. *Annals of Behavioral Medicine*, 53(6):573–582, 2019. doi: 10.1093/abm/kay067. URL https://academic.oup.com/abm/article/53/6/573/5091257.

[434] Guoliang Li, Ruihua Han, Chengyang Li, He Li, Shuai Wang, Wenchao Ding, Hong Zhang, and Chengzhong Xu. Self-evolutionary replanning for failure-aware motion planning, 2026. URL https://arxiv.org/abs/2603.02772.

[435] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering, 2024. URL https://arxiv.org/abs/2405.15793.

[436] Hao Wang, Haocheng Yang, Licheng Pan, Lei Shen, Xiaoxi Li, Yinuo Wang, Zhichao Chen, Yuan Lu, Haoxuan Li, and Zhouchen Lin. Unbiased reward modeling from implicit feedback for LLM alignment, 2026. URL https://arxiv.org/abs/2603.23184.

[437] International Organization for Standardization. *Robots and Robotic Devices: Safety Requirements for Personal Care Robots*. International Organization for Standardization, 2014. URL https://www.iso.org/standard/53820.html.

[438] Li Siyao, Jiawei Gu, Shuai Liu, Kairui Hu, Zekun Li, Linjie Li, Chengcheng Tang, Po-Chen Wu, Ivan Shugurov, Lingni Ma, Michael Zollhoefer, Sizhe An, Abhay Mittal, Amy Zhao, Ranjay Krishna, Manling Li, Ziwei Liu, and Chuan Guo. HumanCLAW: Can Vision-Language Models Act Through a Body?, 2026. URL https://arxiv.org/abs/2607.27180.

[439] Ido Levy, Ben Wiesel, Sami Marreed, Alon Oved, Avi Yaeli, Nir Mashkif, and Segev Shlomov. ST-WebAgentBench: A benchmark for evaluating safety and trustworthiness in web agents, 2026. URL https://arxiv.org/abs/2410.06703.

[440] Yijia Shao, Tianshi Li, Weiyan Shi, Yanchen Liu, and Diyi Yang. PrivacyLens: Evaluating privacy norm awareness of language models in action, 2025. URL https://arxiv.org/abs/2409.00138.

[441] Zeyi Liao, Lingbo Mo, Chejian Xu, Mintong Kang, Jiawei Zhang, Chaowei Xiao, Yuan Tian, Bo Li, and Huan Sun. EIA: Environmental injection attack on generalist web agents for privacy leakage, 2025. URL https://arxiv.org/abs/2409.11295.

[442] Eric Horvitz. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 159–166, 1999.

[443] Daniel C McFarlane. Comparison of four primary methods for coordinating the interruption of people in human-computer interaction. *Human-computer interaction*, 17(1):63–139, 2002.

[444] Shamsi T Iqbal and Brian P Bailey. Effects of intelligent notification management on users and their tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 93–102, 2008.

[445] Haoran Zhang, Luxin Xu, Zhilin Wang, Runquan Gui, Shunkai Zhang, Haodi Lei, Zihao He, Bingsu He, Chicheng Qin, Tong Zhu, Xiaoye Qu, Yang Yang, Yu Cheng, and Yafu Li. π-bench: Evaluating proactive personal assistant agents in long-horizon workflows, 2026. URL https://arxiv.org/abs/2605.14678.

[446] Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Gerard Moreno-Torres Bertran, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Vladislav Vorotilov, Mengjue Wang, Ian Yu, Amine Benhalloum, Grégoire Mialon, and Thomas Scialom. ARE: Scaling up agent environments and evaluations, 2025. URL https://arxiv.org/abs/2509.17158.

[447] Zhenyu Zhao and Roy Zhao. Runtime-independent persistent agents: Preserving identity, memory, and code across models, harnesses, and servers, 2026. URL https://arxiv.org/abs/2609.00546.

[448] Weidi Luo, Shenghong Dai, Xiaogeng Liu, Suman Banerjee, Huan Sun, Muhao Chen, and Chaowei Xiao. AGrail: A lifelong agent guardrail with effective and adaptive safety detection. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8104–8139, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.399. URL https://aclanthology.org/2025.acl-long.399/.

[449] Tommaso Cerruti, Mika Okamoto, and Ansel Kaplan Erol. Agent Memory Is a Surface for Endogenous Authorization Laundering, 2026. URL https://arxiv.org/abs/2609.01836.

[450] Siddhant Arora, Zhiyun Lu, Chung-Cheng Chiu, Ruoming Pang, and Shinji Watanabe. Talking turns: Benchmarking audio foundation models on turn-taking dynamics, 2025. URL https://arxiv.org/abs/2503.01174.

[451] Yuxin Li, Donghang Wu, Guan-Ting Lin, Hung yi Lee, Chengwei Qin, Zhehuai Chen, and Chen Chen. Decoupling conversational dynamics in full-duplex spoken models through reinforcement learning, 2026. URL https://arxiv.org/abs/2607.07148.

[452] Han Luo, Bingbing Wen, and Lucy Lu Wang. Agentic Abstention: Do Agents Know When to Stop Instead of Act?, 2026. URL https://arxiv.org/abs/2606.28733.

[453] Bin Wu, Guanyun Zou, Bingbing Wang, Huan Zhao, and Chuan Shi. Ask now, use later: Benchmarking the proactivity gap in long-lived LLM agents, 2026. URL https://arxiv.org/abs/2605.28108.

[454] Zhao-Yu Zhang, Qingying Chen, Chunyuan Zheng, Jing Zhou, Jian Sun, Siqi Chen, Leiying Chen, Chuan Zhou, Huiyou Jiang, Xin Tao, Haoxuan Li, and Zhouchen Lin. A self-triggered agentic push recommendation system, 2026. URL https://arxiv.org/abs/2608.01949.

[455] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

[456] Ningning Wang, Xavier Hu, Pai Liu, He Zhu, Yue Hou, Heyuan Huang, Shengyu Zhang, Jian Yang, Jiaheng Liu, Ge Zhang, Changwang Zhang, Jun Wang, Yuchen Eleanor Jiang, and Wangchunshu Zhou. Efficient Agents: Building Effective Agents While Reducing Cost, 2025. URL https://arxiv.org/abs/2508.02694.

[457] Ziyu Ma, Hailang Huang, Shun Zou, Yong Wang, Shidong Yang, Yiming Hu, Fei Wei, and XiangXiang Chu. LongHorizon-Harness: Advancing Long-Horizon Agents for Real-World Tasks, 2026. URL https://arxiv.org/abs/2608.01964.

[458] Hongjin Su, Shizhe Diao, Ximing Lu, Mingjie Liu, Jiacheng Xu, Xin Dong, Yonggan Fu, Peter Belcak, Hanrong Ye, Hongxu Yin, Yi Dong, Evelina Bakhturina, Tao Yu, Yejin Choi, Jan Kautz, and Pavlo Molchanov. ToolOrchestra: Elevating Intelligence via Efficient Model and Tool Orchestration, 2025. URL https://arxiv.org/abs/2511.21689.

[459] Wajo. Unveiling Wajo, August 2026. URL https://wajo.ai/blog/unveiling-wajo/. Product blog.

[460] Tatiana Likhomanenko, Richard He Bai, Zijin Gu, Zakaria Aldeneh, Shiladitya Dutta, Luke Carlson, Han Tran, Yizhe Zhang, Ruixiang Zhang, Huangjie Zheng, and Navdeep Jaitly. ChipChat: Low-latency cascaded conversational agent in MLX, 2026. URL https://arxiv.org/abs/2509.00078.

[461] Nils Palumbo, Sarthak Choudhary, Jihye Choi, Guy Amir, Prasad Chalasani, and Somesh Jha. Formal policy enforcement for real-world agentic systems, 2026. URL https://arxiv.org/abs/2602.16708.

[462] Itay Nakash, George Kour, Koren Lazar, Matan Vetzler, Guy Uziel, and Ateret Anaby-Tavor. Effective Red-Teaming of Policy-Adherent Agents, 2025. URL https://arxiv.org/abs/2506.09600.

[463] Qiuyang Zhan, Rui Zhang, Sheng Guo, Lepeng Zhao, and Zhuotao Liu. When memory becomes authority: Benchmarking authority collapse at the memory consolidation boundary, 2026. URL https://arxiv.org/abs/2608.01679.

[464] Yechao Zhang, Shiqian Zhao, Jie Zhang, Gelei Deng, Jiawen Zhang, Xiaogeng Liu, Chaowei Xiao, and Tianwei Zhang. Mind your HEARTBEAT! claw background execution inherently enables silent memory pollution, 2026. URL https://arxiv.org/abs/2603.23064.

[465] Igor Santos-Grueiro. Temporary authority, permanent effects: Commit-time authorization for LLM agents, 2026. URL https://arxiv.org/abs/2607.10487.

[466] Xinbei Ma, Jiyang Qiu, Yao Yao, Zheng Wu, Yijie Lu, Xiangmou Qu, Jiaxin Yin, Xingyu Lou, Jun Wang, Weiwen Liu, Weinan Zhang, Zhuosheng Zhang, and Hai Zhao. Communication policy evolution for proactive LLM agents, 2026. URL https://arxiv.org/abs/2606.14314.

[467] Ziyun Zeng, Hang Hua, Bocheng Zou, Mu Cai, Rogerio Feris, and Jiebo Luo. MementoGUI: Learning agentic multimodal memory control for long-horizon GUI agents, 2026. URL https://arxiv.org/abs/2605.18652.

[468] Zhifei Xie, Zongzheng Hu, Fangda Ye, Xin Zhang, Haobo Chai, Zihang Liu, Pengcheng Wu, Guibin Zhang, Yue Liao, Xiaobin Hu, Deheng Ye, Chunyan Miao, and Shuicheng Yan. PASK: Toward intent-aware proactive agents with long-term memory, 2026. URL https://arxiv.org/abs/2604.08000.

[469] Yibo Lyu, Gongwei Chen, Rui Shao, Weili Guan, and Liqiang Nie. Personalalign: Hierarchical implicit intent alignment for personalized gui agent with long-term user-centric records, 2026. URL https://arxiv.org/abs/2601.09636.

[470] Shenglai Zeng, Xiao Yang, Tiffany Ma, Xinyuan Zhang, Nanshu Wang, Deren Lei, Kai Sun, Xilun Chen, Tammy Stark, Xianhui Zhu, Yuetai Li, Rulin Shao, Tianxin Wei, Rakesh Wanga, Anuj Kumar, Jiliang Tang, Wen-tau Yih, and Xin Luna Dong. MemoProAgent: Toward memory-driven proactive assistance, 2026. URL https://arxiv.org/abs/2610.22671.

[471] Kangning Zhang, Shuai Shao, Qingyao Li, Jianghao Lin, Lingyue Fu, Shijian Wang, Wenxiang Jiao, Yuan Lu, Weiwen Liu, Weinan Zhang, and Yong Yu. MMSkills: Towards Multimodal Skills for General Visual Agents, 2026. URL https://arxiv.org/abs/2605.13527.

[472] Zhiyuan Jiang, Fangrui Huang, Hanwen Xing, Xander Wu, Yipeng Gao, Rui Cao, Mengdi Wang, Shilong Liu, and Yijiang Li. Demystifying Agent Skills: Why They Work-Until They Don't, 2026. URL https://arxiv.org/abs/2608.14036.

[473] Yoojin Kim, Jihyoung Jang, and Hyounghun Kim. PACE: Towards Surfacing Hidden Conflicts in User Requests, 2026. URL https://arxiv.org/abs/2609.03293.

[474] Yumeng Wang, Yuchen Wu, Cheng Qian, Zhiyuan Fan, Hyeonjeong Ha, Shujin Wu, Jiayu Liu, Heng Ji, and Ge Wang. Evaluating the Hidden Costs of Personalization in Large Language Models, 2026. URL https://arxiv.org/abs/2608.28833.

[475] Leyi Pan, Zheyu Fu, Yunpeng Zhai, Shuchang Tao, Sheng Guan, Shiyu Huang, Lingzhe Zhang, Zhaoyang Liu, Bolin Ding, Felix Henry, Aiwei Liu, and Lijie Wen. Omni-SafetyBench: A benchmark for safety evaluation of audio-visual large language models, 2026. URL https://arxiv.org/abs/2508.07173.

[476] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web, 2023. URL https://arxiv.org/abs/2306.06070.

[477] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents, 2025. URL https://arxiv.org/abs/2308.03688.

[478] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents, 2024. URL https://arxiv.org/abs/2307.13854.

[479] Christopher Rawles et al. AndroidWorld: A dynamic benchmarking environment for autonomous agents. *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2405.14573.

[480] Zijian Wu, Xiangyan Liu, Xinyuan Zhang, Lingjun Chen, Fanqing Meng, Lingxiao Du, Yiran Zhao, Fanshi Zhang, Yaoqi Ye, Jiawei Wang, Zirui Wang, Jinjie Ni, Yufan Yang, Arvin Xu, and Michael Qizhe Shieh. MCPMark: A Benchmark for Stress-Testing Realistic and Comprehensive MCP Use, 2025. URL https://arxiv.org/abs/2509.24002.

[481] Ziyang Luo, Zhiqi Shen, Wenzhuo Yang, Zirui Zhao, Prathyusha Jwalapuram, Amrita Saha, Doyen Sahoo, Silvio Savarese, Caiming Xiong, and Junnan Li. MCP-Universe: Benchmarking Large Language Models with Real-World Model Context Protocol Servers, 2025. URL https://arxiv.org/abs/2508.14704.

[482] Yuxuan Zhang, Yubo Wang, Yipeng Zhu, Penghui Du, Junwen Miao, Xuan Lu, Zhuofeng Li, Xingwei Qu, Zhengkang Guo, Yuanzhe Shen, Dingjie Song, Han Zhou, Tuney Zheng, Xian Wu, Hao Yu, Songcheng Cai, Yi Lu, Yunzhuo Hao, Minyi Lei, Liang Chen, Kai Zou, Huifeng Yin, Wendong Xu, Dongfu Jiang, Ping Nie, Jiaheng Liu, Wenhu Chen, and Kelsey R. Allen. ClawBench: Can AI Agents Complete Everyday Online Tasks?, 2026. URL https://arxiv.org/abs/2604.08523.

[483] Chengzhi Liu, Yichen Guo, Yepeng Liu, Yuzhe Yang, Qianqi Yan, Xuandong Zhao, Wenyue Hua, Sheng Liu, Sharon Li, Yuheng Bu, and Xin Eric Wang. Auditing Agent Harness Safety, 2026. URL https://arxiv.org/abs/2605.14271.

[484] Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, Qi Liu, Zhifang Sui, and Tong Yang. Claw-Eval: Towards Trustworthy Evaluation of Autonomous Agents, 2026. URL https://arxiv.org/abs/2604.06132.

[485] Guan-Ting Lin, Chen Chen, Zhehuai Chen, and Hung yi Lee. Full-Duplex-Bench-v3: Benchmarking tool use for full-duplex voice agents under real-world disfluency, 2026. URL https://arxiv.org/abs/2604.04847.

[486] Kuan Li, Shuo Zhang, Huacan Wang, Fangzhou Yu, Zecheng Sheng, Yi Gu, Weipeng Ming, Lei Xue, Chen Liu, Sen Hu, Ronghao Chen, Siyue Lin, Yuqing Hou, Xiaofeng Mou, and Yi Xu. SMH-Bench: Benchmarking LLM agents for environment-grounded reasoning and action in smart homes, 2026. URL https://arxiv.org/abs/2606.01912.

[487] Chen Chen and Zhehuai Chen. JarvisBench: Always-on intelligence between humans and agents, 2026. URL https://arxiv.org/abs/2608.14870.

[488] Ruoyu Zhang, Zhaocen Liu, Weiyi Tian, Melissa Feng, and Shilong Liu. OptionWorld: A benchmark for diagnosing failures of LLMs as long-term personal assistants, 2026. URL https://arxiv.org/abs/2610.28491.

[489] David Pawlan. Assistant benchmark: A public scorecard for AI assistants you can text, 2026. URL https://assistantbenchmark.com/. Version 0.2.

[490] Ke Xu, Yuhao Wang, and Yu Wang. From reactive to proactive: Assessing the proactivity of voice agents via ProVoice-Bench, 2026. URL https://arxiv.org/abs/2604.15037.

[491] Yuanbo Tang, Huaze Tang, Tingyu Cao, Lam Nguyen, Anping Zhang, Xinwen Cao, Chunkang Liu, Wenbo Ding, and Yang Li. ProAgentBench: Evaluating LLM agents for proactive assistance with real-world data, 2026. URL https://arxiv.org/abs/2602.04482.

[492] Nghi D. Q. Bui and Georgios Evangelopoulos. Agentic coding needs proactivity, not just autonomy, 2026. URL https://arxiv.org/abs/2605.06717.

[493] Gil Pasternak, Dheeraj Rajagopal, Julia White, Dhruv Atreja, Matthew Thomas, George Hurn-Maloney, and Ash Lewis. Beyond reactivity: Measuring proactive problem solving in LLM agents, 2026. URL https://arxiv.org/abs/2510.19771.

[494] Manasa Bharadwaj, Yolanda Liu, InJung Yang, Sungil Kim, Nikhil Verma, KoKeun Kim, Kevin Ferreira, and YoungJoon Kim. PersonalHomeBench: Evaluating agents in personalized smart homes, 2026. URL https://arxiv.org/abs/2604.16813.

[495] Lingjun Mao, Jinglin Cao, Lukun He, Wenpeng Xu, Yuchen Yan, Hengyu Jin, Zijun Zhang, Jinzhou Tang, Songyao Jin, Ziche Liu, Yifei Shao, Junbo Huang, Fang Nan, Ruobing Han, Ziqiao Xi, Ziming Xu, Shuang Liang, Sibo Zhu, Wenyi Wu, Xinyue Wang, Yiming Huang, Xinkai Zou, Gao Zhu, Kun Zhou, and Biwei Huang. RobotAgentBench: Towards open-world robot autonomy, 2026. URL https://arxiv.org/abs/2610.34649.

[496] Taewon Yun, Hyeonseong Park, Jeonghwan Choi, Hayoon Park, Yeeun Choi, and Hwanjun Song. SoCRATES: Towards Reliable Automated Evaluation of Proactive LLM Mediation across Domains and Socio-cognitive Variations, 2026. URL https://arxiv.org/abs/2606.05563.

[497] Mengyu Zheng, Kai Han, Boxun Li, Haiyang Xu, Yuchuan Tian, Wei He, Hang Zhou, Jianyuan Guo, Hailin Hu, Lin Ma, Chao Xu, Guohao Dai, Lixue Xia, Yunchao Wei, Yunhe Wang, and Yu Wang. Claw-SWE-Bench: A Benchmark for Evaluating OpenClaw-style Agent Harnesses on Coding Tasks, 2026. URL https://arxiv.org/abs/2606.12344.

[498] Junho Kim, Hosu Lee, James M. Rehg, Minsu Kim, and Yong Man Ro. STRIDE: When to speak meets sequence denoising for streaming video understanding, 2026. URL https://arxiv.org/abs/2603.27593.

[499] Xudong Lu, Yang Bo, Jinpeng Chen, Shuhan Li, Xintong Guo, Huankang Guan, Fang Liu, Dunyuan Xu, Peiwen Sun, Heyang Sun, Rui Liu, and Hongsheng Li. AURA: Always-on understanding and real-time assistance via video streams, 2026. URL https://arxiv.org/abs/2604.04184.

[500] Jinbiao Wei, Qianran Ma, Yilun Zhao, Xiao Zhou, Kangqi Ni, Guo Gan, and Arman Cohan. OpenComputer: Verifiable Software Worlds for Computer-Use Agents, 2026. URL https://arxiv.org/abs/2605.19769.

[501] Shiyun Zhao, Xinwei Song, Tianyu Guo, Xiaomeng Gao, Mingyuan Liu, Xu Han, Yuanyuan Zhang, Zhenliang Zhang, Xue Feng, and Bo Dai. NormAct: Benchmarking embodied agents' proactive compliance with unspoken social norms, 2026. URL https://arxiv.org/abs/2606.27826.

[502] Ruixiang Zhao, Jie Yang, Zijie Xin, Tianyi Wang, Fengyun Rao, Jing LYU, and Xirong Li. OmniPro: A comprehensive benchmark for omni-proactive streaming video understanding, 2026. URL https://arxiv.org/abs/2605.18577.

[503] Zhiyue Xu, Xukai Wang, Zhiyuan Huang, Mingjie Zhan, Jian Zhao, and Xu-Yao Zhang. After the attacker leaves: Benchmarking carryover attacks on LLM agents, 2026. URL https://arxiv.org/abs/2610.28425.

[504] H. W. Jung, Y. Kim, H. Kim, M. K. Kim, H. Lee, J. Y. Park, W. J. Kim, and J. Park. Socially assistive robot Hyodol for depressive symptoms of community-dwelling older adults in medically underserved areas: A preliminary study. *Journal of Clinical Medicine*, 15(1):217, 2026. doi: 10.3390/jcm15010217.

[505] Yichi Zhang, Xin Luna Dong, Zhaojiang Lin, Andrea Madotto, Anuj Kumar, Babak Damavandi, Joyce Chai, and Seungwhan Moon. Proactive assistant dialogue generation from streaming egocentric videos. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 12055–12079, 2025.

[506] Dohyun Lee, Minho Park, Jeonghoon Park, Dongyoon Hwang, Junha Hyung, Sungwon Hwang, Byungkun Lee, and Jaegul Choo. An agentic framework for proactive robots with procedural assistance memory, 2026. URL https://arxiv.org/abs/2610.21211.

[507] Junlong Li, Huaiyuan Xu, Sijie Cheng, Kejun Wu, Kim-Hui Yap, Lap-Pui Chau, and Yi Wang. Building egocentric procedural AI assistant: Methods, benchmarks, and challenges, 2026. URL https://arxiv.org/abs/2511.13261.

[508] Haokai Pang, Yuqian Fu, Yifei Wang, Yujiao Shen, Tianwen Qian, Mohamed Elhoseiny, and Hongbo Fu. EgoProbe: Benchmarking long-range and exact memory in streaming egocentric videos, 2026. URL https://arxiv.org/abs/2610.17128.

[509] Peng Wang, Songshuo Lu, Yaohua Tang, Sijie Yan, Wei Xia, and Yuanjun Xiong. A full-duplex speech dialogue scheme based on large language model. *Advances in Neural Information Processing Systems*, 37: 13372–13403, 2024.

[510] Bandhav Veluri, Benjamin N Peloquin, Bokai Yu, Hongyu Gong, and Shyamnath Gollakota. Beyond turn-based interfaces: Synchronous llms as full-duplex dialogue agents. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 21390–21402, 2024.

[511] Muhammad Umair, Vasanth Sarathy, and Jan Ruiter. Large language models know what to say but not when to speak. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 15503–15514, 2024.

[512] Donghyuk Kim, Sejeong Yang, Wonjin Shin, and Joo-Young Kim. V-Rex: Real-time streaming video LLM acceleration via dynamic KV cache retrieval, 2025. URL https://arxiv.org/abs/2512.12284.

[513] Haoyang Fan, Dhruv Parikh, Rajgopal Kannan, and Viktor Prasanna. StreamAhead: Real-time streaming video question answering via adaptive precomputation, 2026. URL https://arxiv.org/abs/2610.25551.

[514] Mingkang Dong, Muxin Pu, Jie Li, Bohan Guo, Songruo Chen, Bin Ren, Xu Zheng, Chen Zhao, Tianwen Qian, Mohamed Elhoseiny, and Yuqian Fu. ObjectStream: Latent objects as memory anchors for streaming video understanding, 2026. URL https://arxiv.org/abs/2607.28312.

[515] Xudong Lu, Xueying Li, Annan Wang, Yang Bo, Jinpeng Chen, Zengliang Li, Nianzu Yang, Rui Liu, Xue Yang, Jingwen Hou, and Hongsheng Li. OmniInteract: Benchmarking real-world streaming interaction for real-time omnimodal assistants, 2026. URL https://arxiv.org/abs/2605.26485.

[516] Xudong Lu, Huankang Guan, Yang Bo, Jinpeng Chen, Xintong Guo, Shuhan Li, Fang Liu, Peiwen Sun, Xueying Li, Wei Zhang, Xue Yang, Rui Liu, and Hongsheng Li. PhoStream: Benchmarking real-world streaming for omnimodal assistants in mobile scenarios, 2026. URL https://arxiv.org/abs/2601.22575.

[517] Wenhao Zou, Yuwei Miao, Zhanyu Ma, Jun Xu, Jiuchong Gao, Jinghua Hao, Renqing He, and Jingwen Xu. LTS-VoiceAgent: A listen-think-speak framework for efficient streaming voice interaction via semantic triggering and incremental reasoning, 2026. URL https://arxiv.org/abs/2601.19952.

[518] Yongheng Zhang, Ziang Liu, Jiaxuan Zhu, Shuai Wang, Xiangqi Chen, Haojing Huang, Jiayi Kuang, Siyu Chen, Ao Shen, Hao Wu, Qiufeng Wang, Qian-Wen Zhang, Junnan Dong, Wenhao Jiang, Ying Shen, Hai-Tao Zheng, Yinghui Li, Di Yin, Xing Sun, and Philip S. Yu. From Chatbot to Digital Colleague: The Paradigm Shift Toward Persistent Autonomous AI, 2026. URL https://arxiv.org/abs/2606.14502.

[519] Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory, 2026. URL https://arxiv.org/abs/2509.25140.

[520] Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models, 2026. URL https://arxiv.org/abs/2510.04618.

[521] Zhimin Fan, Hongwei Yu, Yeqing Shen, Haolong Yan, Guozhen Peng, Tianhao Peng, Yudong Zhang, Xiaowen Zhang, Kaijun Tan, Zheng Ge, Xiangyu Zhang, and Daxin Jiang. Skill-guided continuation distillation for GUI agents, 2026. URL https://arxiv.org/abs/2606.18890.

[522] Yurun Chen, Biao Yi, Keting Yin, and Shengyu Zhang. SkillHarness: Harnessing safe skills for computer-use agents, 2026. URL https://arxiv.org/abs/2606.20636.

[523] Shilong Jin, Lanjun Wang, and Zhuosheng Zhang. SE-GA: Memory-augmented self-evolution for GUI agents, 2026. URL https://arxiv.org/abs/2605.16883.

[524] Weile Chen, Bingchen Miao, Qifan Yu, Wendong Bu, Guoming Wang, Wenqiao Zhang, Shengyu Zhang, Juncheng Li, and Siliang Tang. Learning to adapt: Self-improving web agent via cognitive-aware exploration, 2026. URL https://arxiv.org/abs/2605.31365.

[525] Daniel Beechey, Derek Yuen, Jianheng Liu, Dezhao Luo, Tiantian He, Weilin Luo, Jun Wang, and Kun Shao. Darwin mobile agent: A roadmap for self-evolution, 2026. URL https://arxiv.org/abs/2606.20622.

[526] Yongtong Wu, Shaoyuan Chen, Yinmin Zhong, Rilin Huang, Yixuan Tan, Wentao Zhang, Liyue Zhang, Shangyan Zhou, Yuxuan Liu, Shunfeng Zhou, Mingxing Zhang, Xin Jin, and Panpan Huang. DualPath: Breaking the Storage Bandwidth Bottleneck in Agentic LLM Inference, 2026. URL https://arxiv.org/abs/2602.21548.

[527] Shuo Yang, Xiaoze Fan, Melissa Pan, Haocheng Xi, Zhe Wang, Shanlin Sun, Kurt Keutzer, Song Han, Matei Zaharia, Chenfeng Xu, and Ion Stoica. FreeToken: Efficient Edge-Native MoE Serving with Bandwidth-Adaptive Execution, 2026. URL https://arxiv.org/abs/2608.16157.

[528] Erik Johannes Husom, Arda Goknil, Merve Astekin, Lwin Khin Shar, Andre Kåsen, Sagar Sen, Benedikt Andreas Mithassel, and Ahmet Soylu. Sustainable LLM inference for edge AI: Evaluating quantized LLMs for energy efficiency, output accuracy, and inference latency, 2025. URL https://arxiv.org/abs/2504.03360.

[529] Siyin Wang, Junhao Shi, Senyu Fei, Zhaoyang Fu, Li Ji, Jingjing Gong, and Xipeng Qiu. In-Context World Modeling for Robotic Control, 2026. URL https://arxiv.org/abs/2606.26025.

[530] Shuai Shao, Qihan Ren, Chen Qian, Boyi Wei, Dadi Guo, Jingyi Yang, Xinhao Song, Linfeng Zhang, Weinan Zhang, Dongrui Liu, and Jing Shao. Your agent may misevolve: Emergent risks in self-evolving LLM agents, 2026. URL https://arxiv.org/abs/2509.26354.

[531] Xutao Mao, Liangjie Zhao, Xiang Zheng, and Cong Wang. Practice makes unsafe: Skill misevolution in self-improving LLM agents, 2026. URL https://arxiv.org/abs/2608.12851.

[532] Hao Li, Tongyan Hu, Weidi Luo, Shanghao Shi, Xiaogeng Liu, Aiden Suh, Yulong Cao, Ning Zhang, William Yeoh, Yevgeniy Vorobeychik, and Chaowei Xiao. AutoRedClaw: Automated red-teaming for claw-style agents, 2026. URL https://arxiv.org/abs/2610.27244.

[533] Chenyu Zhou, Huacan Chai, Wenteng Chen, Zihan Guo, Rong Shan, Yuanyi Song, Tianyi Xu, Yingxuan Yang, Aofan Yu, Weiming Zhang, Congming Zheng, Jiachen Zhu, Zeyu Zheng, Zhuosheng Zhang, Xingyu Lou, Changwang Zhang, Zhihui Fu, Jun Wang, Weiwen Liu, Jianghao Lin, and Weinan Zhang. Externalization in LLM Agents: A Unified Review of Memory, Skills, Protocols and Harness Engineering, 2026. URL https://arxiv.org/abs/2604.08224.

[534] Muye Huang, Lingling Zhang, Xingyu Yu, Lei Shi, Zhanyu Ma, Jun Xu, Jiuchong Gao, Jinghua Hao, Renqing He, and Jun Liu. DuplexOmni: Real-time listening, seeing, thinking, and speaking for full-duplex interaction, 2026. URL https://arxiv.org/abs/2606.09186.

[535] Microsoft. Copilot on Windows: Connectors, and document creation begin rolling out to Windows insiders, October 2025. URL https://blogs.windows.com/windows-insider/2025/10/09/copilot-on-windows-connectors-and-document-creation/. Product blog.

[536] Model Context Protocol. What is the Model Context Protocol (MCP)?, 2026. URL https://modelcontextprotocol.io/.

[537] Ben Nassi, Stav Cohen, and Or Yair. Invitation is all you need! Promptware attacks against LLM-Powered assistants in production are practical and dangerous, 2025. URL https://arxiv.org/abs/2508.12175.

[538] Pavan Reddy and Aditya Sanjay Gujral. EchoLeak: The first real-world zero-click prompt injection exploit in a production LLM system. In *AAAI Fall Symposium Series*, 2025. URL https://arxiv.org/abs/2509.10540.

[539] Zhiqiang Wang, Yichao Gao, Yanting Wang, Suyuan Liu, Haifeng Sun, Haoran Cheng, Guanquan Shi, Haohua Du, and Xiangyang Li. MCPTox: A benchmark for tool poisoning attack on real-world MCP servers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026. URL https://arxiv.org/abs/2508.14925.

[540] Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Securing AI agents with privilege control, 2026. URL https://arxiv.org/abs/2504.11703.

[541] Lillian Tsai and Eugene Bagdasarian. Contextual agent security: A policy for every purpose. In *Workshop on Hot Topics in Operating Systems (HotOS)*, 2025. URL https://arxiv.org/abs/2501.17070.

[542] Yuhao Wu, Franziska Roesner, Tadayoshi Kohno, Ning Zhang, and Umar Iqbal. IsolateGPT: An execution isolation architecture for LLM-based agentic systems. In *Network and Distributed System Security Symposium (NDSS)*, 2025. URL https://arxiv.org/abs/2403.04960.

[543] Christoph Bühler, Matteo Biagiola, Luca Di Grazia, and Guido Salvaneschi. AgentBound: Securing execution boundaries of AI agents. In *ACM International Conference on the Foundations of Software Engineering (FSE)*, 2026. URL https://arxiv.org/abs/2510.21236.

[544] Yujia Qiao and DeepSeek Harness Team. Exploring super-agent design: Towards a more autonomous agent system, 2026. URL https://github.com/deepseek-ai/dsh-superagent-prototype. Design article and research prototype.

[545] Yang Xiao, Yusong Sun, Haoyi Wu, Wenyang Hui, Wen Da, Zhaokai Luo, Mu Chuan, Yao Hu, Wenjie Li, and Chengyue Jiang. PILOT in the loop: Live self-improvement for long-horizon agents, 2026. URL https://arxiv.org/abs/2608.26530.

[546] RayNeo. What are the best smart glasses with an AI assistant?, October 2026. URL https://www.rayneo.com/blogs/news/best-smart-glasses-ai-assistant-rayneo-2026. Product blog.

[547] Oura. Oura Advisor, 2026. URL https://support.ouraring.com/hc/en-us/articles/39512345699219-Oura-Advisor. Product blog.

[548] Google. Gemini Daily Brief: Smart summaries for busy minds, 2026. URL https://gemini.google/overview/daily-brief/. Product blog.

[549] Today AI. Today: The AI personal agent for your whole day, 2026. URL https://today.ai/articles/blog/what-is-today. Product blog.

[550] Matthew Jörke, Defne Genç, Valentin Teutschbein, Shardul Sapkota, Sarah Chung, Paul Schmiedmayer, Maria Ines Campero, Abby C. King, Emma Brunskill, and James A. Landay. Bloom: Designing for LLM-augmented behavior change interactions. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, 2026. doi: 10.1145/3772318.3790506. URL https://arxiv.org/abs/2510.05449.

[551] Ruolan Wu, Chun Yu, Xiaole Pan, Yujia Liu, Ningning Zhang, Yue Fu, Yuhan Wang, Zhi Zheng, Li Chen, Qiaolei Jiang, Xuhai Xu, and Yuanchun Shi. MindShift: Leveraging large language models for mental-states-based problematic smartphone use intervention. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, 2024. URL https://arxiv.org/abs/2309.16639.

[552] Echotime Technology. Orilo: Your personal voice agent, 2026. URL https://orilo.live/. Product page.

[553] Google. 4 ways Pixel's Magic Cue can help you save time, 2025. URL https://blog.google/products-and-platforms/devices/pixel/google-pixel-magic-cue-ai-feature/. Product blog.

[554] Sony Group Corporation. aibo ERS-1000 help guide: Getting to know aibo, 2026. URL https://helpguide.sony.net/aibo/ers1000/v1/en-us/contents/TP0001970093.html. Product blog.

[555] Instinct. Instinct, 2026. URL https://instinct.com/. Product blog.

[556] Louis Grenard. Road to 2.0, March 2026. URL https://blog.getleon.ai/road-to-2-0/. Product blog.

[557] Town. Town documentation: Suggestions, 2026. URL https://town.com/docs/features/suggestions. Product blog.

[558] Justinas Mišeikis, Pietro Caroni, Patricia Duchamp, Alina Gasser, Rastislav Marko, Nelija Mišeikienė, Frederik Zwilling, Charles de Castelbajac, Lucas Eicher, Michael Früh, and Hansruedi Früh. Lio – a personal robot assistant for human-robot interaction and care applications. *IEEE Robotics and Automation Letters*, 2020. doi: 10.1109/LRA.2020.3007462. URL https://arxiv.org/abs/2006.09019.

[559] Peng Liao, Kristjan Greenewald, Predrag Klasnja, and Susan Murphy. Personalized HeartSteps: A reinforcement learning algorithm for optimizing physical activity. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(1), 2020. doi: 10.1145/3381007. URL https://arxiv.org/abs/1909.03539.

[560] Roy Zhao and Zhenyu Zhao. Code is the body: Agent-owned software bodies for recursive evolution and descent, 2026. URL https://arxiv.org/abs/2607.28691.

[561] Even Realities. Even G2 AI glasses, 2026. URL https://www.evenrealities.com/ai-glasses. Product blog.

[562] Memoket. Memoket Gem: A lightweight AI wristband to record conversations, 2026. URL https://memoket.ai/. Product page.

[563] Soulog. Soulog Sense, 2026. URL https://www.soulog.com/products/soulogsense. Product page.

[564] MemoMind. MemoMind One, 2026. URL https://www.memo-mind.com/pages/memomind-one. Product page.

[565] SwitchBot. SwitchBot AI MindClip, 2026. URL https://us.switch-bot.com/products/switchbot-ai-mindclip. Product page.

[566] Careflick. Oak1, 2026. URL https://careflick.com/. Product page.

[567] Looki. What is a wearable AI assistant? how context becomes help, 2026. URL https://www.looki.ai/blogs/news/what-is-a-wearable-ai-assistant-how-real-world-context-becomes-help. Product blog.

[568] Apple. watchOS 26 delivers more personalised ways to stay active and connected, 2025. URL https://www.apple.com/uk/newsroom/2025/06/watchos-26-delivers-more-personalized-ways-to-stay-active-and-connected/. Product blog.

[569] Hark. Hark Pro, 2026. URL https://hark.com/. Product page.

[570] Simular. Sai, 2026. URL https://sai.work. Product page.

[571] Samsung. How to use Now Nudge on Samsung Galaxy phone, 2026. URL https://www.samsung.com/us/support/answer/ANS10010355/. Product blog.

[572] vivo. vivo X200 series frequently asked questions, 2025. URL https://www.vivo.com.cn/service/questions/all?categoryId=158&questionId=1744. Product blog.

[573] OPPO. ColorOS 17, 2026. URL https://www.coloros.com/version/coloros17/. Product blog.

[574] Kindroid. Chat features and tools, 2026. URL https://kindroid.ai/v2/docs/chat-features-and-tools/. Product blog.

[575] LIA. LIA: The personal assistant for everyone, 2026. URL https://lia.jeyswork.com/en. Product page.

[576] Specs. SPECS Intelligence, 2026. URL https://www.specs.com/intelligence. Product page.

[577] Violoop. Violoop, 2026. URL https://violoop.ai/. Product page.

[578] Catch. An executive assistant for travel planning, 2026. URL https://www.catchagent.ai/blog/solutions/executive-assistant-travel-planning. Product blog.

[579] Humansongs, Inc. HeyClicky changelog, 2026. URL https://heyclicky.com/changelog. Product blog.

[580] Vellum. Vellum: Your personal intelligence, 2026. URL https://vellum.ai/. Product blog.

[581] Charlie Tritschler. Meet Astro, a home robot unlike any other, September 2021. URL https://www.aboutamazon.com/news/devices/meet-astro-a-home-robot-unlike-any-other. Product blog.

[582] Amazon. Alexa+, 2026. URL https://www.amazon.com/alexa. Product blog.

[583] Casio. Moflin, 2026. URL https://www.casio.com/uk/moflin/. Product blog.

[584] KEYi Technology. LOONA DeskMate launches at CES 2026: An emotion-aware embodied AI workmate, 2026. URL https://keyirobot.com/en-sg/blogs/news/loona-deskmate-launches-at-ces-2026-an-emotion-aware-embodied-ai-workmate. Product blog.

[585] Adam Zweiger, Jyothish Pari, Han Guo, Ekin Akyürek, Yoon Kim, and Pulkit Agrawal. Self-adapting language models, 2025. URL https://arxiv.org/abs/2506.10943.

[586] Ruiyao Xu, Tiankai Yang, and Wei-Chieh Huang. HyperSkill: Self-evolving LLM agents via hypergraph-structured skill memory, 2026. URL https://arxiv.org/abs/2608.16114.

[587] Aodi Wu, Haodong Han, Xubo Luo, Ruisuo Wang, Shan He, and Xue Wan. SpaceMind: A modular and self-evolving embodied vision-language agent framework for autonomous on-orbit servicing, 2026. URL https://arxiv.org/abs/2604.14399.

[588] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

[589] Peidong Wang, Zhiming Ma, Ying Chang, Xufang Luo, Yiqun Zhang, Zihan Wang, Xiaocui Yang, Shi Feng, Yuqing Yang, and Dongsheng Li. Self-evolving embodied agents via skill-harness evolution, 2026. URL https://arxiv.org/abs/2608.11350.

[590] Xin Ding, Liang Mi, Mingzhe Huang, Zixuan Wang, Chao Zhang, Zixu Hao, Fu Chen, Xiangyu Li, Yikai Zheng, Yaoyu Guo, Weijun Wang, Kun Li, Hao Wu, Yunxin Liu, and Ting Cao. Zetta ζ: An Efficient Closed-Loop Embodied Harness for Self-Evolving Physical Intelligence, 2026. URL https://arxiv.org/abs/2608.16590.

[591] Liwei Che, Benlin Liu, Dai Li, Xiangyan Li, Zijian Wu, Changjiang Li, Minghao Guo, Chengzhi Mao, Vladimir Pavlovic, and Ranjay Krishna. LVMH: A meta-optimization harness for long-video agent memory, 2026. URL https://arxiv.org/abs/2610.23904.

[592] Hui Xue and Fan Yang. Rethinking self-evolving agents: Do we still need prescribed optimization pipelines?, 2026. URL https://arxiv.org/abs/2608.09629.

[593] Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, Danny Driess, Michael Equi, Adnan Esmail, Yunhao Fang, Chelsea Finn, Catherine Glossop, Thomas Godden, Ivan Goryachev, Lachy Groom, Hunter Hancock, Karol Hausman, Gashon Hussein, Brian Ichter, Szymon Jakubczak, Rowan Jen, Tim Jones, Ben Katz, Liyiming Ke, Chandra Kuchi, Marinda Lamb, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Yao Lu, Vishnu Mano, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Charvi Sharma, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, Will Stoeckle, Alex Swerdlow, James Tanner, Marcel Torne, Quan Vuong, Anna Walling, Haohuan Wang, Blake Williams, Sukwon Yoo, Lili Yu, Ury Zhilinsky, and Zhiyuan Zhou. $\pi^*_{0.6}$: a VLA that learns from experience, 2025. URL https://arxiv.org/abs/2511.14759.

[594] Wanxin Tian, Shijie Zhang, Kevin Zhang, Xiaowei Chi, Chunkai Fan, Junyu Lu, Yulin Luo, Qiang Zhou, Yiming Zhao, Ning Liu, Siyu Lin, Zhiyuan Qin, Xiaozhu Ju, Shanghang Zhang, and Jian Tang. SEEA-R1: Tree-structured reinforcement fine-tuning for self-evolving embodied agents, 2025. URL https://arxiv.org/abs/2506.21669.

[595] Qixin Zeng, Shuo Zhang, Hongyin Zhang, Renjie Wang, Han Zhao, Libang Zhao, Runze Li, Donglin Wang, and Chao Huang. CRL-VLA: Continual vision-language-action learning, 2026. URL https://arxiv.org/abs/2602.03445.

[596] Yao He, Gan Sun, Wenqi Liang, Fazeng Li, and Yang Cong. Towards human-like physical intelligence: Lifelong vision-language-action learning for robotic manipulation, 2026. URL https://arxiv.org/abs/2607.14852.

[597] Yangbo Wei, Zhen Huang, Shaoqiang Lu, Junhong Qian, Qifan Wang, Chen Wu, and Lei He. SkillSmith: Co-evolving skills and tools for self-improving agent systems, 2026. URL https://arxiv.org/abs/2606.01314.

[598] Jiaqi Liu, Xinyu Ye, Peng Xia, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. EvolveMem: Self-evolving memory architecture via AutoResearch for LLM agents, 2026. URL https://arxiv.org/abs/2605.13941.

[599] Zayx Shawn. PACE: Anytime-valid acceptance tests for self-evolving agents, 2026. URL https://arxiv.org/abs/2606.08106.

[600] Ruofei Ju, Xinrui Wang, Xin Ding, Yifan Yang, Hao Wu, Shiqi Jiang, Qianxi Zhang, Hao Wen, Xiangyu Li, Weijun Wang, Kun Li, Yunxin Liu, Haipeng Dai, Wei Wang, and Ting Cao. EmbodiSkill: Skill-aware reflection for self-evolving embodied agents, 2026. URL https://arxiv.org/abs/2605.10332.

[601] Minhua Lin, Juncheng Wu, Zijun Wang, Zhan Shi, Yisi Sang, Bing He, Zewen Liu, Tianxin Wei, Zongyu Wu, Zhiwei Zhang, Dakuo Wang, Xiang Zhang, Benoit Dumoulin, Cihang Xie, Yuyin Zhou, Suhang Wang, and Hanqing Lu. Harness updating is not harness benefit: Disentangling evolution capabilities in self-evolving LLM agents, 2026. URL https://arxiv.org/abs/2605.30621.

[602] Sunny Panchal, Apratim Bhattacharyya, Guillaume Berger, Antoine Mercier, Cornelius Bohm, Florian Dietrichkeit, Reza Pourreza, Xuanlin Li, Pulkit Madan, Mingu Lee, Mark Todorovich, Ingo Bax, and Roland Memisevic. What to say and when to say it: Live fitness coaching as a testbed for situated interaction, 2026. URL https://arxiv.org/abs/2407.08101.

[603] Xin Wang, Taein Kwon, Mahdi Rad, Bowen Pan, Ishani Chakraborty, Sean Andrist, Dan Bohus, Ashley Feniello, Bugra Tekin, Felipe Vieira Frujeri, Neel Joshi, and Marc Pollefeys. HoloAssist: an egocentric human interaction dataset for interactive AI assistants in the real world, 2023. URL https://arxiv.org/abs/2309.17024.

[604] Yuhao Dong, Shulin Tian, Shuai Liu, Shuangrui Ding, Yuhang Zang, Xiaoyi Dong, Yuhang Cao, Jiaqi Wang, and Ziwei Liu. Demo-ICL: In-context learning for procedural video knowledge acquisition, 2026. URL https://arxiv.org/abs/2602.08439.

[605] Haojian Huang, Zexi Li, Junhao Guo, Yehang Zhang, Wenxuan Peng, Bohan Zhou, Weilin Ruan, Leyi Wu, Chenxu Wang, Jianchong Su, Binghui Xie, Wosong Chen, Yingjie Xu, Tianhao Zhou, Suzeyu Chen, Pukun Zhao, Jiaqi He, Xinyi Li, Runze Li, Peiran Dong, Shaoxiang Dang, Jing Huang, Yingbing Chen, Yifan Chang, Tianyi Zhang, Shiyuan Deng, Haozhi Wang, Yangkai Wei, Wenqian Li, Han Yang, Kaiwen Zhou, Huaping Liu, James Cheng, Rui Shao, Donglin Wang, Yaochu Jin, Jianye Hao, Ying-Cong Chen, and Yinchuan Li. In-Context Learning for Robots: Methods and Applications, 2026. URL https://arxiv.org/abs/2609.36012.

[606] Fangcheng Liu, Yeqing Shen, Anda Cheng, Weishi Mi, Chao Tang, Chenyuan Liu, Yushun Xiang, Tingguang Li, Yong-Lu Li, and Yehui Tang. RoboICL: Embodied in-context learning with GPT-6 Astra, 2026. URL https://arxiv.org/abs/2609.34261.

[607] Letian Fu, Huang Huang, Gaurav Datta, Lawrence Yunliang Chen, William Chung-Ho Panitch, Fangchen Liu, Hui Li, and Ken Goldberg. In-Context Imitation Learning via Next-Token Prediction, 2024. URL https://arxiv.org/abs/2408.15980.

[608] Hongru Hou, Tiehua Mei, Denghui Geng, Jinhui Huang, Ao Xu, Hengrui Chen, Jiaqing Liang, and Deqing Yang. ProRL: Effective Reinforcement Learning for Proactive Recommendation via Rectified Policy Gradient Estimation, 2026. URL https://arxiv.org/abs/2605.28293.

[609] Eugene Bagdasarian, Lillian Tsai, Sahar Abdelnabi, Ananth Balashankar, Borja Balle, Sebastian Benthall, Igor Bilogrevic, Eleanor Birrell, Elie Bursztein, Kamalika Chaudhuri, et al. Open and emergent problems in agentic privacy and security: A contextual angle. Workshop report, Google, 2026. URL https://storage.googleapis.com/gweb-research2023-media/pubtools/1047874.pdf.

[610] OECD.AI. ByteDance AI Smartphone Triggers App Restrictions and User Harms in China. OECD AI Incidents Monitor, incident 2025-12-07-3ee1, 2025. URL https://oecd.ai/en/incidents/2025-12-07-3ee1.

[611] Manus. Introducing Manus 2.0, September 2026. URL https://manus.im/en/blog/introducing-manus-2-0. Product blog.

[612] Diandian Guo, Cong Cao, Fangfang Yuan, Yingqi Wang, Yueshan Wang, and Dakui Wang. Self-authored verification is unreliable in heuristic self-improving agents, 2026. URL https://arxiv.org/abs/2607.24300.

[613] Nikita Kachaev, Mikhail Kolosov, Daniil Zelezetsky, Alexey K. Kovalev, and Aleksandr I. Panov. Don't Blind Your VLA: Aligning Visual Representations for OOD Generalization, 2025. URL https://arxiv.org/abs/2510.25616.

[614] Ye Yu, Xiaopeng Yuan, Haibo Jin, Heming Liu, Yaoning Yu, and Haohan Wang. Do self-evolving agents forget? capability degradation and preservation in lifelong LLM agent adaptation, 2026. URL https://arxiv.org/abs/2605.09315.

[615] Biswa Sengupta. Self-evolving agents with anytime-valid certificates, 2026. URL https://arxiv.org/abs/2607.00871.

[616] Sihang Jiang, Lipeng Ma, Zhonghua Hong, Keyi Wang, Zhiyu Lu, Tengfei Wang, Shisong Chen, Jinghao Zhang, Tianjun Pan, Weijia Li, Jiaqing Liang, and Yanghua Xiao. SEA-Eval: A benchmark for evaluating self-evolving agents beyond episodic assessment, 2026. URL https://arxiv.org/abs/2604.08988.

[617] Yuqian Wu, Zhijie Deng, Wei Chen, Junwei Li, Yutian Jiang, Junle Chen, Zhengjun Huang, Qingxiang Liu, Jing Tang, Jiaheng Wei, and Yuxuan Liang. LifeSide: Benchmarking agents as lifelong digital companions, 2026. URL https://arxiv.org/abs/2606.04660.

[618] Pin Qian, Su Wang, Yihang Chen, Qiaolin Yu, Xiaoyuan Wang, Zhitong Guo, Zhicheng Wang, and Junxian You. Toward user-conditioned evaluation of personal LLM agents under temporal interventions, 2026. URL https://arxiv.org/abs/2607.21635.

[619] Xing Zhang, Yanwei Cui, Guanghui Wang, Ziyuan Li, Wei Qiu, Bing Zhu, and Peiyang He. Library drift: Diagnosing and fixing a silent failure mode in self-evolving LLM skill libraries, 2026. URL https://arxiv.org/abs/2605.19576.

[620] Hyunin Lee, Jinglue Xu, Jeffrey Seely, Donghyun Lee, Matei Zaharia, and Yujin Tang. Recursive harness self-improvement, 2026. URL https://arxiv.org/abs/2607.15524.

[621] Zae Myung Kim, Young-Jun Lee, Seungyeon Jwa, and Dongyeop Kang. Meta^n: Recursive self-improvement through emergent depth, 2026. URL https://arxiv.org/abs/2608.24735.

[622] Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents, 2026. URL https://arxiv.org/abs/2603.19461.

[623] Alex Iacob, Andrej Jovanović, William F. Shen, Daniel Burkhardt, Meghdad Kurmanji, Nurbek Tastan, Lorenzo Sani, Niccolò Alberto Elia Venanzi, Jiayi Nie, Ambroise Odonnat, Zeyu Cao, Bill Marino, Xinchi Qiu, Rika Antonova, and Nicholas D. Lane. The Red Queen Gödel machine: Co-evolving agents and their evaluators, 2026. URL https://arxiv.org/abs/2606.26294.

[624] Yiming Xiong, Shengran Hu, and Jeff Clune. Learning to continually learn via meta-learning agentic memory designs, 2026. URL https://arxiv.org/abs/2602.07755.

[625] Zhanzhi Lou, Hui Chen, Yibo Li, Qian Wang, and Bryan Hooi. Meta-TTL: Meta-learning self-improvement policies for language agents, 2026. URL https://arxiv.org/abs/2604.00830.

[626] Mingguang Chen, Licheng Wang, and Bo Qu. Recursive self-improvement in AI: From bounded self-refinement to autonomous research loops, 2026. URL https://arxiv.org/abs/2607.07663.

[627] Juan Perdomo, Tijana Zrnic, Celestine Mendler-Dünner, and Moritz Hardt. Performative prediction. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 7599–7609, 2020. URL https://proceedings.mlr.press/v119/perdomo20a.html.

[628] Allison J. B. Chaney, Brandon M. Stewart, and Barbara E. Engelhardt. How algorithmic confounding in recommendation systems increases homogeneity and decreases utility. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 224–232, 2018. doi: 10.1145/3240323.3240370. URL https://arxiv.org/abs/1710.11214.

[629] Rulin Shao, Akari Asai, Shannon Zejiang Shen, Hamish Ivison, Varsha Kishore, Jingming Zhuo, Xinran Zhao, Molly Park, Samuel G. Finlayson, David Sontag, Tyler Murray, Sewon Min, Pradeep Dasigi, Luca Soldaini, Faeze Brahman, Wen tau Yih, Tongshuang Wu, Luke Zettlemoyer, Yoon Kim, Hannaneh Hajishirzi, and Pang Wei Koh. DR Tulu: Reinforcement Learning with Evolving Rubrics for Deep Research, 2026. URL https://arxiv.org/abs/2511.19399.

[630] Qiyuan Zhang, Junyi Zhou, Yufei Wang, Fuyuan Lyu, Yidong Ming, Can Xu, Qingfeng Sun, Kai Zheng, Peng Kang, Xue Liu, and Chen Ma. RubricBench: Aligning Model-Generated Rubrics with Human Standards, 2026. URL https://arxiv.org/abs/2603.01562.

[631] Binghai Wang, Chenlong Zhang, Dayiheng Liu, Jiajun Zhang, Jiawei Chen, Mingze Li, Mouxiang Chen, Rongyao Fang, Siyuan Zhang, Xuwu Wang, Yuheng Jing, Zeyao Ma, and Zeyu Cui. The Verification Horizon: No Silver Bullet for Coding Agent Rewards, 2026. URL https://arxiv.org/abs/2606.26300.

[632] Xing Zhang, Guanghui Wang, Yanwei Cui, Ziyuan Li, Wei Qiu, Bing Zhu, and Peiyang He. Who grades the grader? co-evolving evaluation metrics and skills for self-improving LLM agents, 2026. URL https://arxiv.org/abs/2607.12790.