Procedural Assistance Memory for Proactive Robot

Dohyun Lee*, **Minho Park***, **Jeonghoon Park, Dongyoon Hwang, Junha Hyung, Sungwon Hwang, Byungkun Lee, Jaegul Choo**

KAIST AI

{aiclaudev, m.park}@kaist.ac.kr

* indicates equal contribution.

Abstract: Proactive robot assistance requires acting on user needs before explicit requests, but zero-shot vision-language models struggle in this setting: lacking knowledge of the user's personal routine, they treat every situation as general and uncertain, defaulting to inaction rather than risking an incorrect intervention. We observe that everyday routines are highly repetitive, so even a few interactions reveal stable user-specific patterns that can ground proactive decisions. Based on this, we propose a System 1–2 framework in which a VLM task predictor reads a *Procedural Assistance Memory* (PAM) alongside the current observation to decide when and how to assist, and a VLA executor realizes the predicted task as low-level robot action. PAM is structured as a finite-state machine over activity phases, per-state assistance rules, and event-driven transitions; a reflection VLM revises PAM from each day's interaction trace without weight updates or retraining. We evaluate on a multi-day desk-assistance scenario with six pick-and-place tasks. Memory-free baselines remain conservative across all five evaluation days, yielding near-zero recall on proactive tasks. Our system improves consistently with accumulated experience, achieving the highest precision, recall, and F1 and reducing first-fire delay from approximately 5 seconds on Day 1 to 1 second by Day 5. Ablations further show that the FSM structure, rather than memory alone, is the primary driver of these gains.

Keywords: Proactive Robot Assistance, Memory-based VLM, Self-Evolving.
Links: Project page | Github Code

1 Introduction

A truly useful assistant in everyday life does not only respond to requests, but also anticipates when help would be valuable. This may involve handing over an object at the right moment, putting away an item after it has been used, or preparing something that will be needed in the next step. By contrast, robot assistants are still commonly evaluated as systems that execute explicit user commands [1, 2]. For robots to play the role of capable assistants in household, and home-office settings, they must move beyond waiting for instructions and learn to recognize when intervention would be helpful. This has motivated growing interest in *proactive robot assistance*: robots that infer when help may be useful and act in advance of explicit commands.

One line of work in proactive robot assistance predicts task progress in structured sequential tasks [3–6]. However, these approaches are largely confined to robots that perform a single task by following a fixed protocol (e.g., chair assembly), and are ill-suited to generalist robots tasked with open-ended housekeeping. Recently emerging generalist Vision-Language-Action models (VLAs) [7–11] may appear to offer an alternative, but robotics datasets remain scarce across the diverse scenarios such settings demand. As a result, end-to-end models that map perception directly to action face clear limitations even when equipped with reasoning, and proactive tasks, where intent is often underspecified, have received little attention. In response, recent work has adopted a System 1–2 design [12] that explicitly separates a high-level, proactive Vision-Language Model (VLM) from a VLA that executes atomic tasks [13–16]. This separation simplifies learning, since the VLM decides

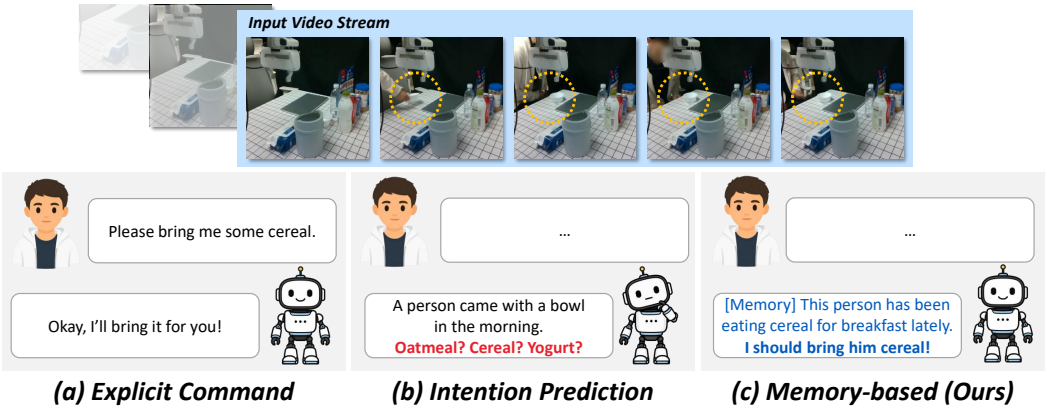

Figure 1: **Motivation for memory-based proactive assistance.** Visually similar observations map to different correct actions depending on user-specific habits, and our PAM memory resolves this ambiguity by conditioning the task predictor on accumulated behavioral context.

when and what to help with while the VLA need only carry out the prescribed action. We adopt this structure, which in turn places the burden of proactivity squarely on the high-level VLM.

The central challenge, then, is accurately predicting human intention at the high level [17], and recent approaches leverage pretrained, large-scale VLMs to do so in open-ended settings [18–20]. Yet zero-shot prediction remains unreliable when the same observable behavior corresponds to different desired outcomes across users: one person setting out a mug wants coffee, another wants hot water for tea (Fig. 1). A fixed, pretrained predictor cannot resolve such ambiguity, because the missing information is not in the scene but in the individual user's habits.

Crucially, this missing information can be learned during deployment. In housekeeping settings, an individual's behavior is highly repetitive, so even a handful of past interactions reveal stable preferences, and corrective feedback gathered once can be reused whenever a similar situation recurs. This suggests that a proactive robot should not treat each deployment as zero-shot, but should accumulate experience about its specific user over time. We therefore investigate a test-time learning approach for the high-level VLM, building a robot that personalizes quickly from few interactions, continues adapting after deployment, and carries user feedback forward into future behavior.

To this end, we introduce *Procedural Assistance Memory* (PAM), a continually updated deployment-time memory that records when assistance is needed and what assistive task should be issued for a particular user. PAM structures this memory as a finite-state machine over the user's activity, where states represent activity phases, transitions represent progress between phases, and per-state assistance entries store the assistive opportunities associated with each phase. In our five-day scenario, the user repeatedly performs a daily activity; PAM is updated across these days so that the robot can provide increasingly appropriate assistance.

Our contributions are threefold:

1. We introduce a **memory-based approach** for proactive robot assistance. By conditioning the high-level VLM on user-specific deployment experience, the system can decide both *when* to assist and *what* task to issue, while retaining the generality of foundation-model reasoning.

2. We propose **Procedural Assistance Memory** (PAM), a finite-state memory that structures user activity into phases, transitions, and per-state assistance entries. This representation makes proactive decisions context-dependent, reducing ambiguity when the same observation can imply different assistance needs across users or activity phases.

3. We develop a **continually updated memory loop** in which a memory reflector revises PAM from observed behavior and user feedback during deployment. This enables proactive assistance to improve over time without retraining, as the robot incorporates new routine elements, changed behavior, and feedback into future task setting.

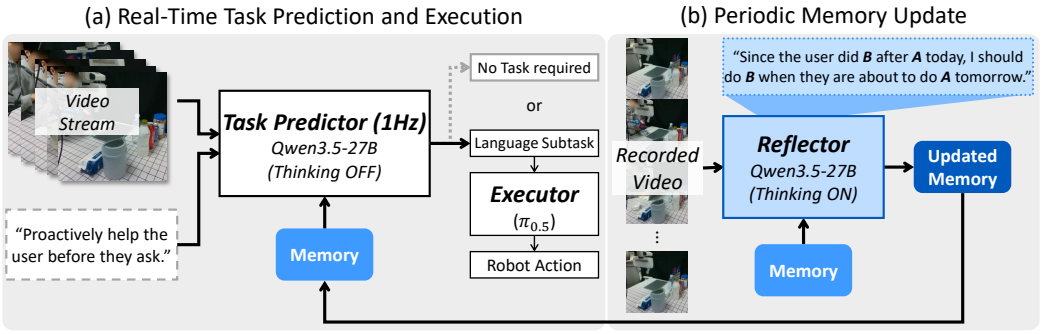

Figure 2: **Overview of our framework.** (a) A VLM task predictor reads the PAM memory and recent frames to set assistive tasks for the VLA executor. (b) At the end of each day, a reflection VLM updates the PAM from the day's video by adding or revising states, rules, and transitions.

2 Related Work

2.1 Vision-Language-Action Models

Generalist VLAs [7–11] combine large-scale visual-language pretraining with action prediction to generalize across tasks, with subsequent work augmenting them with chain-of-thought reasoning [21–23], spatial grounding [24, 25], and world-model predictions [26–29]. However, jointly optimizing action execution and high-level reasoning in a single network causes the two objectives to compete under limited robot data. In response, recent work adopts a System 1–2 design [12] that explicitly separates a slow reasoning module from a fast action executor [9, 13–16], and we build on this structure in our framework.

2.2 Proactive Assistance in Embodied HRI

A prominent line of work on proactive robots studies assistance in structured sequential tasks, where the human activity is already organized into known subtasks. In these settings, robots estimate task progress or subtask completion to provide help at the right moment [5, 6], such as handovers in assembly [3] or ingredient delivery in cooking [4]. This direction is especially effective when the task sequence, progress measure, and assistance opportunities are known in advance. Another line of work models object usage patterns in daily environments to anticipate future assistance needs [30–32]. These methods move beyond single structured tasks, but they typically require substantial interaction logs or curated observations for training, limiting their flexibility across diverse everyday routines and rapidly changing user behavior.

With the recent progress of foundation models, proactive assistance has begun to move toward more general intent inference. Some systems rely on non-verbal cues such as visual scene changes [18], human-object interactions [33, 34], or gaze [35, 36] to infer assistive intent, while others draw on verbal cues such as indirect instructions [20], speech [19], or dialogue [37, 38]. However, intent inference from momentary cues alone remains ambiguous: the same cue can imply different assistance needs depending on the user, context, and activity history. We address this by conditioning the proactive decision on a user-specific memory accumulated across days, so that assistance is grounded not only in the current cue but also in how this user has typically acted and been assisted.

2.3 Evolving Memory for Test-Time Adaptation

Recent LLM-agent systems use external memory to support test-time adaptation without weight updates, allowing an agent to accumulate and reuse experience across episodes. These works span verbal self-reflection [39], reusable skill libraries [40], workflow or experience extraction [41, 42], and long-term personalized stores [43, 44]. However, their memories are generally designed for text-centric agents, games, or instruction-following tasks, rather than for deciding when a robot

should proactively assist during an evolving human activity. PAM differs in both content and use: it stores assistive opportunities, organizes them into activity phases, per-state assistance entries, and transitions, and provides the context used by a task predictor to decide when assistance is needed and what task should be issued.

3 Method

Our framework consists of two components, shown in Fig. 2. First, instead of a monolithic VLA, we adopt an explicitly separated System 1–2 architecture that divides proactive task prediction and execution, reducing the difficulty of each sub-problem. Second, we introduce a periodic memory update module that supplies an Procedural Assistance Memory (PAM) to the task predictor, which substantially reduces the ambiguity inherent in proactive task prediction (Fig. 1). We detail each component in the following sections.

3.1 Real-Time Task Prediction and Execution via System 1–2 Architecture

Since our setting is fully proactive and the user issues no task-specific instructions, the task predictor takes as input only the most recent W video frames together with a fixed prompt, *"Proactively help the user before they ask."* From these inputs, the predictor decides what, if anything, to do next. When the user is, for example, pouring cereal into a bowl, the predictor reasons that milk should be brought next and emits a language subtask such as *"bring milk."* When nothing needs to be done, it emits `NO_ACT`. Any non-`NO_ACT` subtask is then passed to a VLA executor that specializes in atomic skills, which converts it into low-level robot actions.

Reasoning purely from the current observation, however, is insufficient whenever the user's intention is ambiguous, as already discussed in Section 1. To resolve such ambiguity, we condition the task predictor on an additional memory that encodes the user's routine; the next section describes how this memory is automatically constructed and updated.

3.2 Periodic Update of Procedural Assistance Memory (PAM)

Finite-State-Machine (FSM) Memory. The simplest design would be to store experience as a flat list of trigger-action pairs, but such a representation easily loses the overall flow of a routine: each pair is interpreted in isolation, with no notion of what has already happened or what should come next. We therefore organize the memory as a Finite-State-Machine (FSM), where each state aggregates the trigger-action pairs that are relevant within it, and transitions encode how the routine progresses from one state to another. An example is shown in Fig. 3. At inference time, we prompt the predictor to first identify which state of the FSM the current observation belongs to, and then to decide its action by consulting the trigger-action pairs and outgoing transitions attached to that state. This two-step reasoning lets the predictor exploit both the immediate trigger and the surrounding routine context.

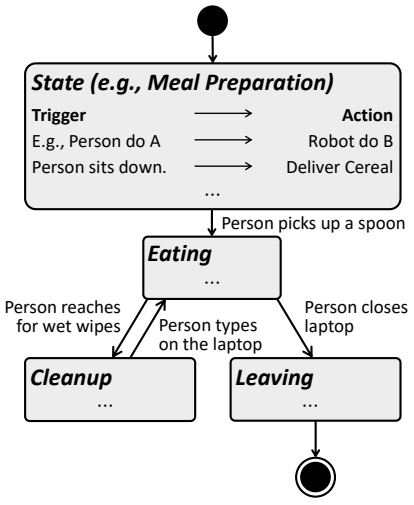

Figure 3: **An example of our proposed PAM memory**, comprising states, trigger–action pairs, and transitions.

Periodic Memory Update. The PAM memory is constructed and refined automatically. Following the self-evolving framework of [39, 40], at the end of each day we feed the day's video together with the memory used during that day into a reasoning-enabled VLM, which we refer to as the *reflector*, that proposes updates to the PAM. As illustrated in Fig. 2(b), the video contains segments in which the user performed actions that the robot did not assist with. The reflector inspects these segments to identify opportunities the robot could have covered, infers the

Figure 4: **Tasks and dataset.** The six proactive tasks form a continuous desk-activity sequence: (T1) Cereal→Person, (T2) Milk→Person, (T3) Cereal→Storage, (T4) Milk→Storage, (T5) Bowl→Storage, (T6) Used wet wipe→Trash.

underlying user habit (e.g., *"when this user brings out an empty bowl, they pour cereal into it"*), and writes the corresponding trigger-action pair into the appropriate state, so that the robot can act on it the following day. The same procedure may also introduce new states, modify existing transitions, or revise prior trigger-action pairs, gradually personalizing the PAM to the user. While the update frequency is a design choice, we run it once per day, at the end of the full scenario.

4 Experiments

In the following sections, we aim to resolve the following research questions:

RQ1. Does our PAM improve proactive task prediction over memory-free baselines (zero-shot, event-driven, progress-based)? (Section 4.2)

RQ2. Does the proactive predictor benefit from accumulated experience across days, and is the FSM structure essential to this benefit? (Section 4.2)

RQ3. Does the explicit System 1–2 decomposition improve task success and reduce undesired actions compared to monolithic VLAs ($\pi_{0.5}$ with and without reasoning)? (Section 4.3)

RQ4. How does the memory evolve across days, and can the reflector adapt the PAM when the user's routine changes? (Section 4.4)

4.1 Experimental Setup

Tasks. We prepare six proactive tasks that form a continuous sequence (Fig. 4): (T1) deliver cereal (Cereal→Person), (T2) carry over milk (Milk→Person), (T3) place back cereal (Cereal→Storage), (T4) place back milk (Milk→Storage), (T5) place back the empty bowl (Bowl→Storage), and (T6) throw away the used wet wipe (Used wet wipe→Trash). We collect 10 teleoperation demonstrations of the entire scenario and label the correct task frame-by-frame.

Evaluation protocol. We formalize proactive task prediction as a per-frame classification problem. Importantly, the label set comprises not only the six tasks above but also a `NO_ACT` class indicating that no intervention is warranted. We report precision, recall, and F1 score per class (the six proactive tasks and `NO_ACT`) along with their macro averages. We also report the first-fire delay, measured from when a proactive task becomes appropriate to when the model first predicts it. Missed predictions default to the user's own completion time from the demonstrations. To assess the effectiveness of our PAM, we conduct a multi-day evaluation over 5 days, in which the PAM persists across days and is updated from feedback collected on previous days.

Implementation details. We use Qwen 3.5 [45] as the VLM for both the task predictor and the reflector. The reflector runs with thinking mode enabled over a uniform budget of 200 frames per day, while the task predictor runs with thinking mode disabled and produces structured JSON output over a per-tick window of $W=5$ frames sampled at stride 2 from the 1 fps stream. The two stages take approximately 1 minute and 1 second per call, respectively. For the executor, we finetune $\pi_{0.5}$ [11] on our downstream demonstrations. We collect 10 demonstrations per task which is fairly matched to

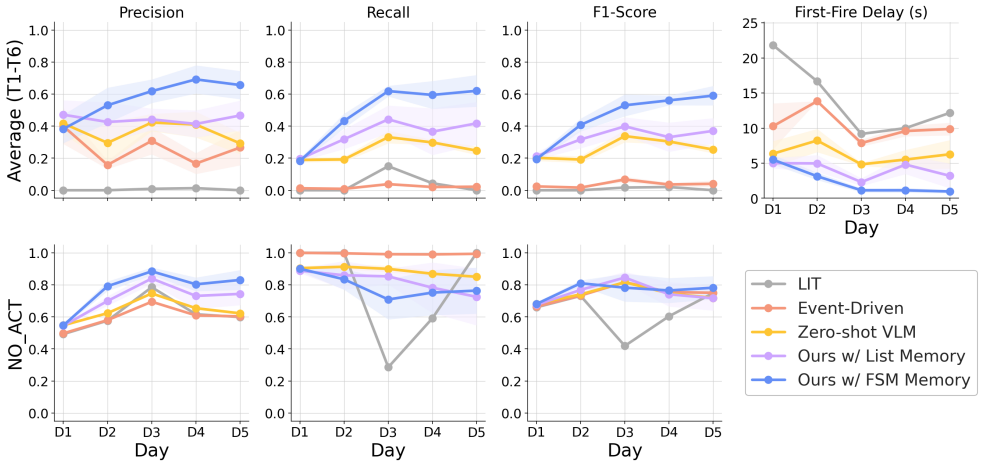

Figure 5: **Multi-day proactive task prediction (Days 1–5).** Memory-free baselines remain flat with near-zero T1–T6 recall, while Ours (FSM Memory) consistently improves across days, achieving the highest F1 and reducing first-fire delay from $\sim$5 s on Day 1 to $\sim$1 s by Day 5.

all baselines with LeRobot framework [46, 47]. Sampling hyperparameters are fixed across all conditions and reported in the supplementary material. Source code and dataset will be made publicly available.

Baselines. For the proactive task prediction baseline, we provide following:

- **LIT** [4]. Builds a task graph over detected objects and estimates the current progress along the graph to predict the next task. This represents structured, progress-tracking proactivity.
- **Event-driven VLM** [18]. Predicts the next task only when a scene change is detected via optical flow, by comparing the pre- and post-change frames. This represents reactive, change-triggered proactivity without any cross-day memory.
- **Zero-shot VLM** [45]. A state-of-the-art VLM that predicts the next proactive task directly from the current observation, without any adaptation. This isolates the benefit of our test-time memory against a strong but static predictor.
- **Ours w/ List Memory**. A variant of our framework that replaces the FSM-structured memory with a flat list of trigger-action pairs accumulated over deployment, while keeping the rest of the pipeline identical. This isolates the contribution of the FSM structure itself.

For a fair comparison, we implement the baselines with Qwen 3.5 [45]. Additionally, we provide two VLA baselines to compare the final task success rate and false-alarm rate on the proactive tasks:

- $\pi_{0.5}$ [11]. The state-of-the-art open-source pretrained VLA. We prompt it with "Proactively help the user before they ask." at both training and test time throughout the entire scenario.
- $\pi_{0.5}$ **(Reasoning)** [11]. Since $\pi_{0.5}$ can decompose a high-level instruction into low-level atomic actions, we finetune it with "Proactively help the user before they ask." as the high-level prompt and the corresponding atomic command (*e.g.*, "pick the cereal and place it next to the user") as the low-level task, leveraging its reasoning capability.

4.2 VLM Proactiveness across Memory Designs

Effectiveness of Memory. As shown in Fig. 5, baselines without memory (LIT [4], Event-Driven [18], Zero-shot VLM [45]) predict conservatively: lacking prior knowledge of the user's routine, they default to `NO_ACT` whenever the observation is ambiguous, which inflates `NO_ACT` recall but suppresses proactiveness on T1–T6. LIT and Event-Driven rarely fire any proactive task, yielding near-zero recall, and the Zero-shot VLM plateaus from Day 2 onward with no mechanism to internalize recurring patterns. All three remain essentially flat across the five days, confirming that

Table 1: **Task-wise success rate comparison with monolithic VLA** [11] with and without subtask reasoning. Bold = best per task.

Method	Cereal→Person	Milk→Person	Cereal→Storage	Milk→Storage	Bowl→Storage	Wet Wipe→Trash
$\pi_{0.5}$ [11]	7 / 10	1 / 10	6 / 10	0 / 10	5 / 10	4 / 10
$\pi_{0.5}$ (Reasoning)	6 / 10	0 / 10	0 / 10	0 / 10	5 / 10	5 / 10
Ours	**9 / 10**	**3 / 10**	**8 / 10**	**5 / 10**	**8 / 10**	**8 / 10**

Figure 6: **Qualitative comparison of closed-loop execution.** Our System 1–2 framework correctly selects and executes proactive tasks, while $\pi_{0.5}$ and $\pi_{0.5}$ (Reasoning) produce incorrect actions due to joint optimization of reasoning and low-level control under limited data.

without memory there is no path from repeated exposure to better proactive behavior. In contrast, our memory-based variants (Ours w/ List, Ours w/ FSM) leverage repeated exposure to improve over the five days, though to different degrees, as we analyze next.

Ablation of Memory Design. Among the two memory designs, Ours w/ List Memory saturates early, since the flat list offers no abstraction over recurring states as it grows. In contrast, Ours w/ FSM Memory consistently improves across days, achieving the highest precision, recall, and F1 on T1–T6 while reducing first-fire delay from roughly 5 seconds on Day 1 to about 1 second by Day 5. This shows that the gains come from the FSM structure itself, not simply from adding memory. The trade-off is that FSM Memory's eagerness to act lowers its `NO_ACT` recall relative to the conservative baselines, although a correspondingly higher `NO_ACT` precision keeps its `NO_ACT` F1 on par with the others. Reducing false alarms further while preserving proactiveness remains an open direction. Per-task results are in the supplementary material.

4.3 Comparison with Monolithic VLAs

As shown in Tab. 1 and Fig. 6, $\pi_{0.5}$, with or without reasoning, suffers from lower success rates on the atomic tasks themselves, since the end-to-end model is forced to jointly learn robot actions and VLM-style reasoning. For example, Milk→Storage in Fig. 6 is among the harder atomic skills, as it requires grasping the rounded milk cap; $\pi_{0.5}$ trained with a single prompt across all tasks lacks the capacity to specialize on such individual skills, dragging down per-task success.

This degradation is especially pronounced in the reasoning variant under our low-data regime. Even though we provide $\pi_{0.5}$ (Reasoning) with frame-level supervision specifying which language subtask should be executed, the model still frequently produces incorrect reasoning labels at inference time. For instance, when the correct prediction is Cereal→Storage but the user happens to be holding the milk, the visible milk often misleads the model into reasoning Milk→Storage, and such reasoning errors in turn make the generated action even more likely to fail. In contrast, our framework delegates atomic action generation to the strong pretrained $\pi_{0.5}$ backbone and offloads reasoning to Qwen 3.5, achieving higher accuracy on both axes.

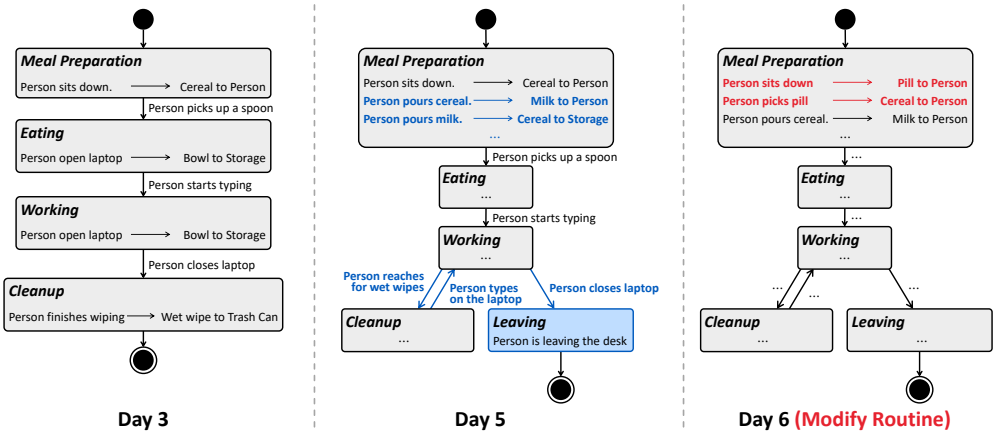

Figure 7: **Qualitative analysis of FSM-based Procedural Assistance Memory evolution.** From Day 3–5, missing trigger-action pairs are recovered and per-state rules are completed. When the user's routine changes, end-of-day reflection revises the affected transitions and rules accordingly.

4.4 Memory Analysis

We qualitatively investigate the FSM-based Procedural Assistance Memory (PAM) in Fig. 7, which highlights both the efficiency and flexibility of our memory structure. Comparing Day 3 and Day 5, the FSM-based PAM becomes increasingly accurate: trigger-action pairs that were missing on Day 3 are recovered on Day 5, with each state capturing its full set of triggers and actions (*e.g.*, the pair "person pours cereal" $\rightarrow$ "milk to person" is added). Our FSM-based PAM also adapts dynamically when the user's routine changes. For instance, even when the user sits down with a bowl as before, taking medication before eating the cereal that the robot has just delivered causes the transition "person sits down" $\rightarrow$ "cereal to person" to be revised, after reflection, to "person sits down" $\rightarrow$ "pill to person". The full raw FSM memory and the reflector's reasoning traces are provided in the supplementary material.

5 Conclusion

We studied proactive robot assistance as a longitudinal personalization problem, in which a robot accumulates user-specific experience across deployment rather than treating each day as zero-shot. We introduced Procedural Assistance Memory (PAM), a finite-state machine over activity phases, per-state assistance rules, and event-driven transitions; a high-level VLM reads PAM at inference time, and a separate reflection VLM updates it from each day's interaction trace without weight updates or retraining. Coupled with a System 1–2 decomposition that offloads low-level execution to a VLA, this design consistently improves proactive precision, recall, and F1 over five evaluation days, reduces first-fire delay from roughly 5 seconds on Day 1 to about 1 second by Day 5, and adapts the FSM to introduced and re-ordered routine elements. Ablations indicate that the FSM structure, rather than memory alone, is the main driver of these gains. We view structured, self-evolving memory as a practical path toward generalist robot assistants that personalize quickly and continue to adapt after deployment.

Limitation. Two failure modes recur in our analysis. First, FSM Memory occasionally fires when the user briefly handles a learned trigger object without committing to that step, which lowers its `NO_ACT` recall (Fig. 5); in such genuinely ambiguous situations the right behavior is often to ask the user rather than to act, but soliciting clarification lies outside our current framework. Second, due to the limited capacity of current open-source VLAs, our evaluation is restricted to seen scenarios rather than novel objects, layouts, or users; recent advances in generalist VLAs [48–54] suggest that scaling to broader scenarios is an increasingly feasible next step.

References

[1] T. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *RSS*, 2023. 1

[2] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *IJRR*, 2023. 1

[3] D. De Lazzari, M. Terreran, G. Giacomuzzo, S. Jain, P. Falco, R. Carli, and D. Romeres. PACE: Proactive assistance in human-robot collaboration through action-completion estimation. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025. doi:10.1109/ICRA55743.2025.11127399. 1, 3

[4] Z. Huang, J. Pohovey, A. Yammanuru, and K. Driggs-Campbell. LIT: Large language model driven intention tracking for proactive human-robot collaboration — a robot sous-chef application. *arXiv preprint arXiv:2406.13787*, 2024. 3, 6, 17

[5] R. Liu, R. Chen, A. Abuduweili, and C. Liu. Proactive human-robot co-assembly: Leveraging human intention prediction and robust safe control. In *2023 IEEE Conference on Control Technology and Applications (CCTA)*, pages 339–345. IEEE, 2023. 3

[6] T. Stouraitis and M. Gienger. Predictive and robust robot assistance for sequential manipulation. *IEEE Robotics and Automation Letters*, 8(12):8026–8033, 2023. 1, 3

[7] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, Q. Vuong, T. Kollar, B. Burchfiel, R. Tedrake, D. Sadigh, S. Levine, P. Liang, and C. Finn. OpenVLA: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024. 1, 3

[8] O. M. Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, T. Kreiman, C. Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

[9] NVIDIA, J. Bjorck, F. Castañeda, N. Cherniadev, X. Da, R. Ding, L. Fan, Y. Fang, D. Fox, F. Hu, et al. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025. 3

[10] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. π_0: A vision-language-action flow model for general robot control. In *Robotics: Science and Systems (RSS)*, 2025.

[11] Physical Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, M. Y. Galliker, D. Ghosh, L. Groom, K. Hausman, B. Ichter, S. Jakubczak, T. Jones, L. Ke, D. LeBlanc, S. Levine, A. Li-Bell, M. Mothukuri, S. Nair, K. Pertsch, A. Z. Ren, L. X. Shi, L. Smith, J. T. Springenberg, K. Stachowicz, J. Tanner, Q. Vuong, H. Walke, A. Walling, H. Wang, L. Yu, and U. Zhilinsky. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025. 1, 3, 5, 6, 7, 15, 16

[12] D. Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, New York, 2011. 1, 3

[13] C. Cui, P. Ding, W. Song, S. Bai, X. Tong, Z. Ge, R. Suo, W. Zhou, Y. Liu, B. Jia, et al. Openhelix: A short survey, empirical analysis, and open-source dual-system vla model for robotic manipulation. *arXiv preprint arXiv:2505.03912*, 2025. 1, 3

[14] H. Song, D. Qu, Y. Yao, Q. Chen, Q. Lv, Y. Tang, M. Shi, G. Ren, M. Yao, B. Zhao, et al. Hume: Introducing system-2 thinking in visual-language-action model. *arXiv preprint arXiv:2505.21432*, 2025.

[15] H. Chen, J. Liu, C. Gu, Z. Liu, R. Zhang, X. Li, X. He, Y. Guo, C.-W. Fu, S. Zhang, et al. Fast-in-slow: A dual-system foundation model unifying fast manipulation within slow reasoning. *arXiv preprint arXiv:2506.01953*, 2025.

[16] B. Han, J. Kim, and J. Jang. A dual process vla: Efficient robotic manipulation leveraging vlm. *arXiv preprint arXiv:2410.15549*, 2024. 1, 3

[17] S. Buyukgoz, J. Grosinger, M. Chetouani, and A. Saffiotti. Two ways to make your robot proactive: Reasoning about human intentions or reasoning about possible futures. *Frontiers in Robotics and AI*, 9:929267, 2022. 2

[18] F. Liu, H. Su, H. Chi, R. Geng, C. Ren, X. Liu, Y. Xu, Y. Ohsita, and L. Zhang. Event-driven proactive assistive manipulation with grounded vision-language planning. *arXiv preprint arXiv:2603.23950*, 2026. 2, 3, 6, 17

[19] S. Wang, J. Fu, F. Liu, X. He, H. Wu, J. Shi, K. Huang, Z. Fei, J. Gong, Z. Wu, Y.-G. Jiang, S.-K. Ng, T.-S. Chua, and X. Qiu. RoboOmni: Proactive robot manipulation in omni-modal context. *arXiv preprint arXiv:2510.23763*, 2025. 3

[20] Y. Chen, K. Gu, Y. Wen, Y. Zhao, T. Wang, and L. Nie. IntentionVLA: Generalizable and efficient embodied intention reasoning for human-robot interaction. *arXiv preprint arXiv:2510.07778*, 2025. 2, 3

[21] M. Zawalski, W. Chen, K. Pertsch, O. Mees, C. Finn, and S. Levine. Robotic control via embodied chain-of-thought reasoning. *arXiv preprint arXiv:2407.08693*, 2024. 3

[22] Q. Zhao, Y. Lu, M. J. Kim, Z. Fu, Z. Zhang, Y. Wu, Z. Li, Q. Ma, et al. CoT-VLA: Visual chain-of-thought reasoning for vision-language-action models. *arXiv preprint arXiv:2503.22020*, 2025.

[23] S. Belkhale, T. Ding, T. Xiao, P. Sermanet, Q. Vuong, J. Tompson, Y. Chebotar, D. Dwibedi, and D. Sadigh. RT-H: Action hierarchies using language. *arXiv preprint arXiv:2403.01823*, 2024. 3

[24] H. Zhen, X. Qiu, P. Chen, J. Yang, X. Yan, Y. Du, Y. Hong, and C. Gan. 3D-VLA: A 3D vision-language-action generative world model. *arXiv preprint arXiv:2403.09631*, 2024. 3

[25] D. Qu, H. Song, Q. Chen, Y. Yao, X. Ye, Y. Ding, Z. Wang, J. Gu, et al. SpatialVLA: Exploring spatial representations for visual-language-action model. *arXiv preprint arXiv:2501.15830*, 2025. 3

[26] Y. Hu, Y. Guo, P. Wang, X. Chen, Y.-J. Wang, J. Zhang, K. Sreenath, and C. Lu. Video prediction policy: A generalist robot policy with predictive visual representations. *arXiv preprint arXiv:2412.14803*, 2024. 3

[27] W. Zhang, H. Liu, Z. Qi, Y. Wang, X. Yu, J. Zhang, R. Dong, J. He, et al. DreamVLA: A vision-language-action model dreamed with comprehensive world knowledge. *arXiv preprint arXiv:2507.04447*, 2025.

[28] J. Ye, S. Gong, J. Gao, J. Fan, S. Wu, W. Bi, H. Bai, L. Shang, et al. Dream-VL & Dream-VLA: Open vision-language and vision-language-action models with diffusion language model backbone. *arXiv preprint arXiv:2512.22615*, 2025.

[29] Y. Tian, Y. Jin, B. Yu, Y. Shi, H. Wu, C. H. Liu, K. Chen, and C. Huang. STARRY: Spatial-temporal action-centric world modeling for robotic manipulation. *arXiv preprint arXiv:2604.26848*, 2026. 3

[30] M. Patel and S. Chernova. Proactive robot assistance via spatio-temporal object modeling. In *Conference on Robot Learning (CoRL)*, 2022. 3

[31] M. Patel, A. Prakash, and S. Chernova. Predicting routine object usage for proactive robot assistance. In *Conference on Robot Learning (CoRL)*, 2023.

[32] M. Patel and S. Chernova. Longitudinal proactive robot assistance. In *Companion of the 2023 ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. ACM, 2023. doi:10.1145/3568294.3579982. 3

[33] E. V. Mascaro, D. Sliwowski, and D. Lee. HOI4ABOT: Human-object interaction anticipation for human intention reading collaborative roBOTs. In *Conference on Robot Learning (CoRL)*, 2024. 3

[34] H. Rahimi, C. Grislain, A. J. Cretides, O. Sigaud, and M. Chetouani. Intentvlm: Open-vocabulary intention recognition through forward-inverse modeling with video-language models. *arXiv preprint arXiv:2604.24002*, 2026. 3

[35] T. Y. H. Tay, X. Yan, J. Ouyang, D. Wu, W. Jiang, J. Kao, and Y. Cui. Intent at a glance: Gaze-guided robotic manipulation via foundation models. *arXiv preprint arXiv:2601.05336*, 2026. 3

[36] J. A. Gaus, W. Ilg, and D. Haeufle. When to act: Calibrated confidence for reliable human intention prediction in assistive robotics. *arXiv preprint arXiv:2601.04982*, 2026. 3

[37] G. H. Sarch, B. T. Kumaravel, S. Ravi, V. Vineet, and A. D. Wilson. Grounding task assistance with multimodal cues from a single demonstration. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 12807–12833, 2025. 3

[38] G. Sarch, Y. Wu, M. J. Tarr, and K. Fragkiadaki. HELPER: Open-ended instructable embodied agents with memory-augmented large language models. *arXiv preprint arXiv:2310.15127*, 2023. 3

[39] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. 3, 4

[40] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. 3, 4

[41] A. Zhao, D. Huang, Q. Xu, M. Lin, Y.-J. Liu, and G. Huang. ExpeL: LLM agents are experiential learners. In *AAAI Conference on Artificial Intelligence*, 2024. 3

[42] Z. Z. Wang, J. Mao, D. Fried, and G. Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024. 3

[43] W. Zhong, L. Guo, Q. Gao, H. Ye, and Y. Wang. MemoryBank: Enhancing large language models with long-term memory. In *AAAI Conference on Artificial Intelligence*, 2024. 3

[44] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Annual ACM Symposium on User Interface Software and Technology (UIST)*, 2023. 3

[45] Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL `https://qwen.ai/blog?id=qwen3.5`. 5, 6, 14, 17

[46] R. Cadene, S. Alibert, A. Soare, Q. Gallouedec, A. Zouitine, S. Palma, P. Kooijmans, M. Aractingi, M. Shukor, D. Aubakirova, M. Russi, F. Capuano, C. Pascal, J. Choghari, J. Moss, and T. Wolf. Lerobot: State-of-the-art machine learning for real-world robotics in pytorch. `https://github.com/huggingface/lerobot`, 2024. 6

[47] Z. K. Weng, M. L. Elwin, and H. Liu. Levr: A modular vr teleoperation framework for imitation learning in dexterous manipulation. *arXiv preprint arXiv:2509.14349*, 2025. 6

[48] P. Intelligence, A. Amin, R. Aniceto, A. Balakrishna, K. Black, K. Conley, G. Connors, J. Darpinian, K. Dhabalia, J. DiCarlo, et al. $\pi^*_{0.6}$: a vla that learns from experience. *arXiv preprint arXiv:2511.14759*, 2025. 8

[49] P. Intelligence, B. Ai, A. Amin, R. Aniceto, A. Balakrishna, G. Balke, K. Black, G. Bokinsky, S. Cao, T. Charbonnier, et al. $\pi_{0.7}$ a steerable generalist robotic foundation model with emergent capabilities. *arXiv preprint arXiv:2604.15483*, 2026.

[50] G. A. Team. Gen-1: Scaling embodied foundation models to mastery. *Generalist AI Blog*, 2026. https://generalistai.com/blog/apr-02-2026-GEN-1.

[51] D. Kim, H. Jang, M. Koo, S. Jang, T. Kim, B. Kim, B. Yoon, C. Jang, D. Choi, D. Han, et al. Rldx-1 technical report. *arXiv preprint arXiv:2605.03269*, 2026.

[52] S. Liu, B. Li, K. Ma, L. Wu, H. Tan, X. Ouyang, H. Su, and J. Zhu. Rdt2: Exploring the scaling limit of umi data towards zero-shot cross-embodiment generalization. *arXiv preprint arXiv:2602.03310*, 2026.

[53] W. Wu, F. Lu, Y. Wang, S. Yang, S. Liu, F. Wang, Q. Zhu, H. Sun, Y. Wang, S. Ma, et al. A pragmatic vla foundation model. *arXiv preprint arXiv:2601.18692*, 2026.

[54] I. Apanasevich, M. Artemyev, R. Babakyan, P. Fedotova, D. Grankin, E. Kupryashin, A. Misailidi, D. Nerus, A. Nutalapati, G. Sidorov, et al. Green-vla: Staged vision-language-action model for generalist robots. *arXiv preprint arXiv:2602.00919*, 2026. 8

[55] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023. 14

[56] D. Driess, J. Springenberg, B. Ichter, L. Yu, A. Li-Bell, K. Pertsch, A. Ren, H. Walke, Q. Vuong, L. X. Shi, et al. Knowledge insulating vision-language-action models: Train fast, run fast, generalize better. *Advances in Neural Information Processing Systems*, 38:102867–102888, 2026. 16

[57] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 16

[58] S. Nasiriany, A. Maddukuri, L. Zhang, A. Parikh, A. Lo, A. Joshi, A. Mandlekar, and Y. Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. *arXiv preprint arXiv:2406.02523*, 2024. 16

[59] A. Khazatsky, K. Pertsch, S. Nair, A. Balakrishna, S. Dasari, S. Karamcheti, S. Nasiriany, M. K. Srirama, L. Y. Chen, K. Ellis, et al. Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024. 16

[60] H. Xue, J. Ren, W. Chen, G. Zhang, Y. Fang, G. Gu, H. Xu, and C. Lu. Reactive diffusion policy: Slow-fast visual-tactile policy learning for contact-rich manipulation. *arXiv preprint arXiv:2503.02881*, 2025. 16

Appendix

In this supplementary material, we provide additional analyses, experiments, qualitative and quantitative results, and implementation details that were omitted from the main paper due to space limitations. We first describe the detailed experimental setup, including the evaluation dataset and annotation procedure, the evaluation settings, the training and deployment of VLAs, and the hardware configuration, to ensure reproducibility (Section A). We then report the per-task results of VLM proactiveness, complementing the aggregate metrics reported in the main paper (Section B). Next, we present a further analysis of the memory update process, examining the effectiveness of memory evolution, the reasoning traces behind each update, and a comparison across different memory structures (Section C). We further analyze diverse scenarios, showing that our method generalizes to a structurally different routine with substantially weaker intent cues and that it adapts to routine changes when the deployed memory must transfer across scenarios (Section D). Finally, we include the full prompts used for the task predictor (Section E) and the reflector (Section F) across our List Memory and FSM Memory variants. Below is the table of contents for the supplementary material.

A Detailed Experimental Setup

A.1 Evaluation Dataset and Annotation

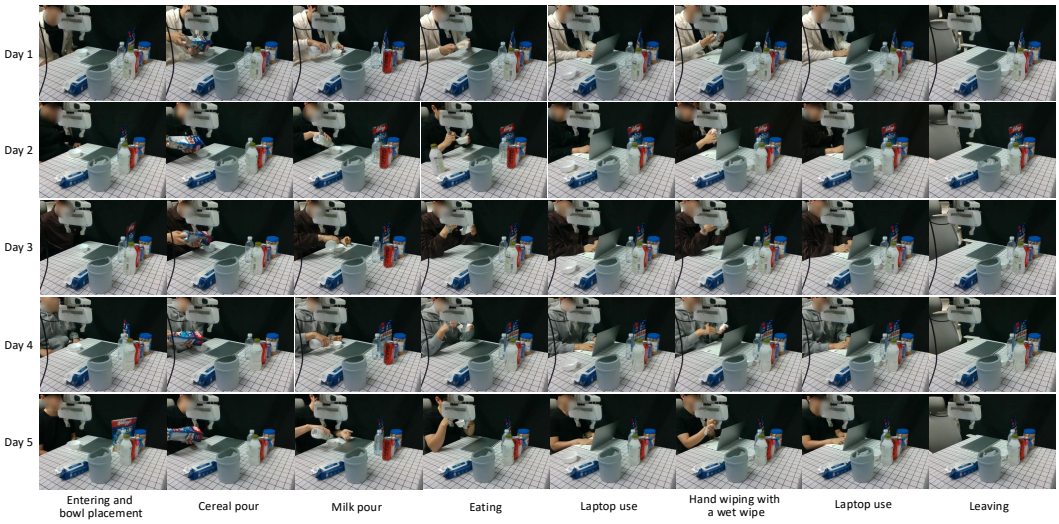

Figure 8: Snapshots of the five evaluation days. Each day captures the same canonical desk routine performed by a single user: bringing out a bowl, pouring cereal and milk, eating, using the laptop with the bowl set aside, wiping the mouth with a wet wipe, and leaving the desk.

To examine how proactive-task prediction performance evolves with repeated exposure to a user's routine, we record five days of video of a single user performing the same canonical desk routine (Fig. 8). Each day follows the same high-level sequence: the user enters the desk with a bowl, pours cereal and then milk into the bowl, eats the meal, sets the empty bowl aside while switching to laptop work, wipes their mouth with a wet wipe and leaves it on the desk, and finally leaves the desk. Within this shared backbone, the precise timing of each step and the visible state of the desk vary naturally across days, so each day is a distinct realization of the same routine rather than a copy. This makes the five-day stream a controlled testbed for asking whether a proactive predictor can convert repeated exposure into faster and more confident task firing, as opposed to treating each day as a fresh zero-shot scene.

Per-frame annotation. We treat proactive-task prediction as a per-frame classification problem and therefore label every evaluation frame. Each day's recording lasts roughly 3.5 to 4.5 minutes; we uniformly sample at 1 fps and obtain 207 to 281 frames per day, for a total of $1{,}200$ annotated frames across the five days (Tab. 2). The authors manually assign one label per sampled frame from the following seven categories: the six proactive tasks T1–T6 (delivering cereal, delivering milk, putting away cereal, putting away milk, putting away the empty bowl, throwing away the used wet wipe), and `NO_ACT` for frames where no intervention is appropriate. A frame receives a task label when, given the current scene and the user's ongoing behavior, that task is the most natural proactive intervention; all other frames are labeled `NO_ACT`. The resulting per-frame labels constitute the ground truth used throughout the experiments in the main paper, and we will release the annotation tool together with the dataset.

A.2 Evaluation Settings

Both the task predictor and the reflector use the same Qwen 3.5–27B backbone [45] served via vLLM [55], but they differ in the input format, the use of thinking, and the call cadence.

Task predictor. At each tick the task predictor receives the most recent $W{=}5$ frames sampled at stride 2 from the 1 fps stream, i.e., the frames at t, $t-2$, $t-4$, $t-6$, $t-8$ seconds, together

Table 2: Per-day annotation statistics on the five evaluation days. Frames are extracted at 1 fps; deleted frames are excluded from the totals. T1: Cereal→Person, T2: Milk→Person, T3: Cereal→Storage, T4: Milk→Storage, T5: Bowl→Storage, T6: Wet wipe→Trash. `NO_ACT` marks frames where no intervention is appropriate.

Day	Frames	NO_ACT	T1	T2	T3	T4	T5	T6
1	281	133	12	21	5	34	35	30
2	261	143	11	27	6	18	24	19
3	234	154	9	23	6	8	14	10
4	217	125	8	23	5	16	16	14
5	207	118	7	19	4	13	21	15
Total	1,200	673	47	113	26	89	110	88

with the current memory and the wall-clock time of the final frame. Each frame is the horizontal concatenation of the two back-facing cameras (left and right; Fig. 9); the wrist camera is not exposed to the VLM stack. Thinking mode is *disabled*, so the model emits the structured JSON {observation, reasoning, task index} directly without an intermediate trace. Sampling is performed at temperature 0.3. Each call takes approximately 1 second on average across all five days. The full system prompts driving this stage, one per memory variant (no memory, list, FSM), are provided in Section E.

Reflector. The reflector is invoked once at the end of each day. Its visual input is the day's recording uniformly downsampled to 200 frames, again using the two-view (left and right) back-facing concatenation without the wrist camera; the prior memory, an action-history summary of the day, and (when provided) user feedback are appended as text. Thinking mode is *enabled*, so the model first produces an explicit reasoning trace before emitting the updated memory JSON (FSM or list schema). Sampling is performed at temperature 0.3. Each call takes approximately 1 minute on average across all five days. The full system prompts for building memory from scratch on the first day and for end-of-day updates on subsequent days, separately for the list and FSM variants, are provided in Section F.

A.3 Training and Deployment for VLAs

All three VLA variants in our study are fine-tuned from the same $\pi_{0.5}$ backbone but differ in what they have to learn. Throughout the paper we consider six atomic proactive tasks: (T1) *Pick up the cereal and place it next to the person.*, (T2) *Pick up the milk and place it next to the person.*, (T3) *Pick up the cereal and place it in the designated storage area.*, (T4) *Pick up the milk and place it in the designated storage area.*, (T5) *Pick up the bowl and place it in the designated storage area.*, and (T6) *Pick up the used wet wipe and place it in the trash can.* All teleoperation demonstrations are recorded on the Franka arm at 20 fps with three RGB streams (two back-facing cameras and one wrist camera; Fig. 9). Unlike the VLM stack, all three VLA variants receive all three views as input, including the wrist camera that provides close-up grasp context.

$\pi_{0.5}$ **(executor) [11].** Our executor only needs to realize one atomic task at a time, dispatched by the task predictor. We therefore collect 10 teleoperation episodes per task and fine-tune $\pi_{0.5}$ on this set, conditioning each episode on its corresponding instruction taken verbatim from T1–T6. At deployment, the executor receives the instruction emitted by the task predictor and reproduces the same input format.

$\pi_{0.5}$ **(monolithic) [11].** The monolithic baseline has no external task predictor and must therefore learn both *when* to act and *what* to do inside a single model. To match this requirement, we collect 10 end-to-end teleoperation episodes of the full scenario, in which the user enters the desk and proceeds through the routine, and the robot, controlled by an expert teleoperator, performs every proactive task at the appropriate moment without any external instruction. We fine-tune $\pi_{0.5}$ on these full-scenario

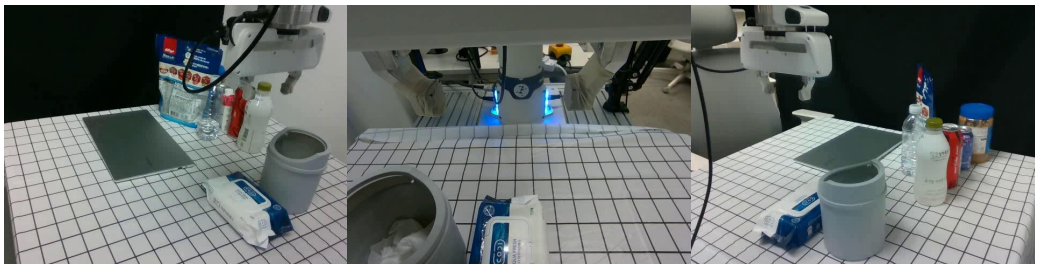

Figure 9: Camera setup. Two back-facing cameras (left and right) observe the user and the desk from behind the Franka arm, while a wrist camera mounted on the end-effector provides close-up grasp context. The VLM-based task predictor and reflector consume only the left and right back views; the VLAs additionally receive the wrist view.

demonstrations, conditioned on a single fixed high-level instruction, *"Actively and proactively help humans before they even ask."*, that does not specify which subtask should be issued at any particular moment.

$\pi_{0.5}$ (monolithic + reasoning) [11, 56]. The reasoning baseline uses the same 10 full-scenario episodes as the monolithic variant, but additionally relies on per-frame subtask supervision so that $\pi_{0.5}$'s native reasoning channel can be trained explicitly [56]. Each frame is annotated with a subtask string, which is either one of T1–T6 or *"No action is required right now."* when no intervention is appropriate. During training, the prompt fed to $\pi_{0.5}$ has the form `Task:{HL}. Subtask:{sub}. State:...;\nAction:` with the same fixed high-level instruction as the monolithic variant, and the total loss combines the standard action-prediction loss with a cross-entropy term on the predicted subtask text (with weight 1.0); at inference time the model first emits the subtask and then conditions its action on its own subtask prediction.

Training and deployment settings. All three VLA variants share the same training and deployment recipe. We fine-tune $\pi_{0.5}$ for 50,000 optimizer steps with AdamW [57] (betas $(0.9, 0.95)$, $\epsilon{=}10^{-8}$, weight decay 0.01, gradient-norm clipping at 1.0). The learning rate is linearly warmed up from 0 to 1×10^{-4} over the first 1,000 steps and then cosine-annealed over the next 30,000 steps to a minimum of 2.5×10^{-6}. Training uses a per-GPU batch size of 16 across 4-B200 GPUs without gradient accumulation, for an effective batch size of 64. The action head predicts chunks of 50 steps (`chunk_size`=50), and at deployment we execute the full predicted chunk before requesting a new one, with 10 denoising steps per chunk (`num_inference_steps`=10).

A.4 Hardware Setup

Robot. We use a Franka Research 3 (FR3) 7-DoF manipulator equipped with the built-in parallel-jaw gripper. Low-level control is provided by the Deoxys real-time controller, which exposes operational-space control of the end-effector. The control loop runs at 20 Hz throughout teleoperation, training, and deployment.

Cameras. Followed by [58, 59], three Intel RealSense cameras are used for both data collection and deployment, all streaming RGB at 30 fps and time-aligned to the 20 Hz control loop. Two cameras (`left_back` and `right_back`) are statically mounted behind the robot at table height, framing the desk and the seated user from two angles, while the third camera is rigidly attached to the wrist and looks down toward the gripper. The two back-facing streams are horizontally concatenated to form the input given to the VLM stack, whereas the VLA variants additionally consume the wrist stream as a third image channel.

Teleoperation interface. Teleoperation demonstrations are collected with TactAR [60], a Meta Quest 3-based VR teleoperation interface. The operator wears the Quest headset and uses the right

hand-held controller; controller pose updates are streamed over HTTP to a FastAPI server, converted from the Unity coordinate frame to the robot base frame, and applied as absolute target end-effector poses with 1:1 translation scaling. Gripper open/close is bound to a controller trigger. The same setup is used for all three VLA training datasets (executor, monolithic, and monolithic + reasoning).

B Per-task Results of VLM Proactiveness

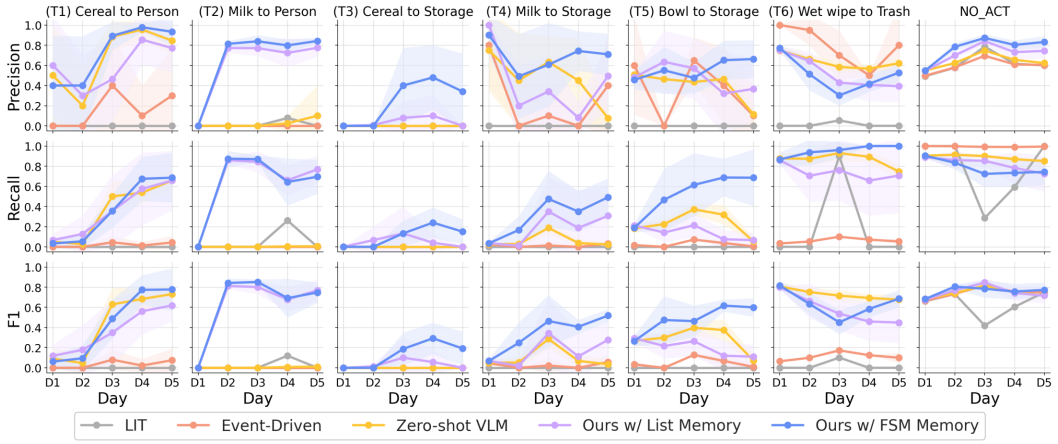

Figure 10: Per-task Precision, Recall, and F1 score comparison. Memory-based approaches especially outperform Milk→Person, Cereal→Storage, while maintaining no action performance as shown in F1 score. In addition, our FSM memory especially outperforms at Cereal→Storage, Milk→Storage.

Memory-augmented variants improve across days on every task. For both memory variants, per-task precision, recall, and F1 rise from Day 1 to Day 5 across all of T1–T6, while the memory-free baselines (LIT [4], Event-driven [18], and Zero-shot VLM [45]) stay flat (Fig. 10). The cross-day gains reported in the main paper thus reflect a broad improvement across the routine rather than a few easy tasks: as the reflector accumulates trigger-action pairs over successive days, each task's rules become more complete and better timed.

Memory matters most where preference disambiguates action. The gap over memory-free baselines is largest on T2 (Milk→Person) and T3 (Cereal→Storage), where the correct action cannot be read from the current scene alone: what to bring after the user pours cereal depends on this user's habitual pairing, and when to put the cereal bag away depends on the routine's typical progression, both available only through accumulated experience. Where the immediate cue is already strong (T1, an empty bowl; T6, finishing wiping), the gap narrows, since even memory-free models can read it.

FSM structure helps most on storage actions. Among the memory variants, FSM Memory's lead over List Memory is largest on the three put-away actions, T3 (Cereal→Storage), T4 (Milk→Storage), and T5 (Bowl→Storage). Each must fire at a precise moment, after its item is finished with but before the user moves on, and FSM Memory places each rule inside the phase where that item is used, so its trigger is interpreted only within the relevant routine context. List Memory matches the same trigger globally, where it is easily confused with similar moments elsewhere, leading to mistimed firing. This phase-local firing also curbs false alarms: FSM Memory's eagerness slightly lowers its `NO_ACT` recall, but its higher `NO_ACT` precision keeps `NO_ACT` F1 on par with the conservative baselines (Section 4.2).

C Detailed Analysis of Memory Update

C.1 Effectiveness of Memory Evolution

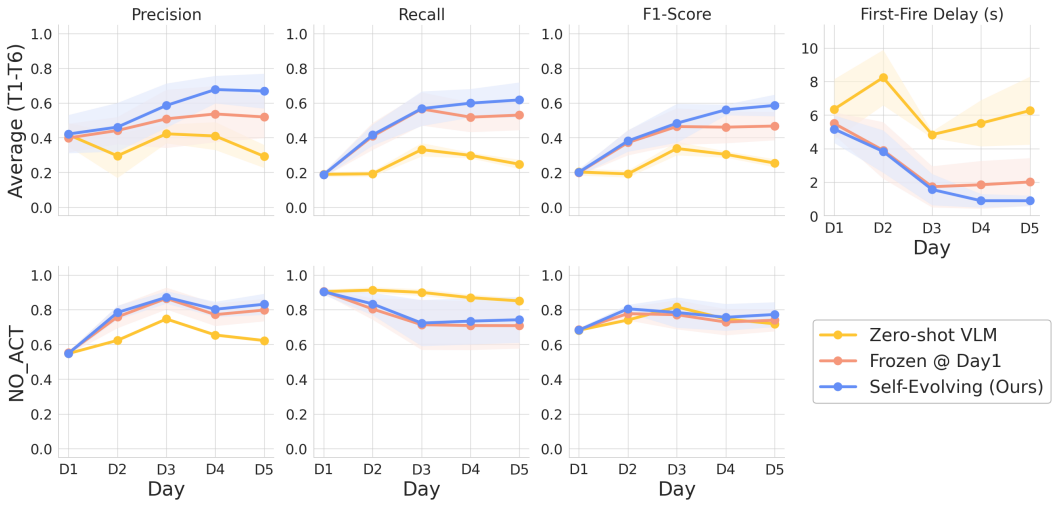

Figure 11: **Effect of memory self-evolution.** Per-day precision, recall, and F1 averaged over the six proactive tasks (top) and on `NO_ACT` (bottom), with first-fire delay (top right), across the five days. We compare our self-evolving memory against a memory frozen after Day 1 and a memory-free zero-shot VLM; only the self-evolving variant keeps improving through Day 5.

This experiment isolates the contribution of the self-evolving update itself. All three conditions share the same inference module and differ only in how the memory changes over time: *Self-Evolving (Ours)* refines the memory with the reflector at the end of each day, *Frozen @ Day1* builds the memory once and applies that Day-1 snapshot unchanged thereafter, and *Zero-shot VLM* carries no memory at all.

Continued reflection is what sustains improvement. On T1–T6, our self-evolving memory improves steadily in precision, recall, and F1 from Day 1 to Day 5 (Fig. 11). The frozen memory gains over the zero-shot VLM once its memory is in use, but from then on it stops improving on its own: its movement closely mirrors the zero-shot curve, which means its day-to-day variation does not come from any further memory refinement. Our self-evolving memory instead pulls steadily away from both, and the gap over the frozen memory widens across days, showing that most of the cross-day gain comes from continued reflection rather than from merely having a memory. The zero-shot VLM stays lowest throughout, confirming that repeated exposure alone yields no improvement without a mechanism to retain it.

Faster and better-timed assistance. The same trend appears in first-fire delay: our self-evolving memory reduces the delay from roughly 5 seconds on Day 1 to about 1 second by Day 5, while the frozen memory plateaus near 2 seconds and the zero-shot VLM stays high and erratic. As the reflector adds and sharpens triggers, the robot not only fires on the right tasks but fires earlier within each routine. On `NO_ACT`, the three conditions remain comparable, so these gains in proactiveness do not come at the cost of over-acting.

Qualitative Analysis of Memory Update Fig. 12 traces the FSM memory after each day's reflection for one run, culminating in the instance shown in Fig. 13. Day 1 installs the skeleton: four states (`meal_preparation` → `eating` → `working` → `leaving`) with two core pairs, bring-cereal and bowl-storage. Day 2 adds the bring-milk pair to `meal_preparation` and broadens the bring-cereal trigger with an *or*-clause. Day 3 adds the put-cereal-back and put-milk-back pairs to the same state, and broadens the bring-milk and bowl-storage triggers. Day 4 introduces a new `cleaning_break`

state with the wet-wipe pair and a `working`↔`cleaning_break` loop, and further broadens both put-away triggers. Day 5 introduces no new pairs or states; the reflector only finishes broadening the remaining triggers, so the trigger lines grow while the actions and FSM topology stay fixed, yielding the final memory in Fig. 13.

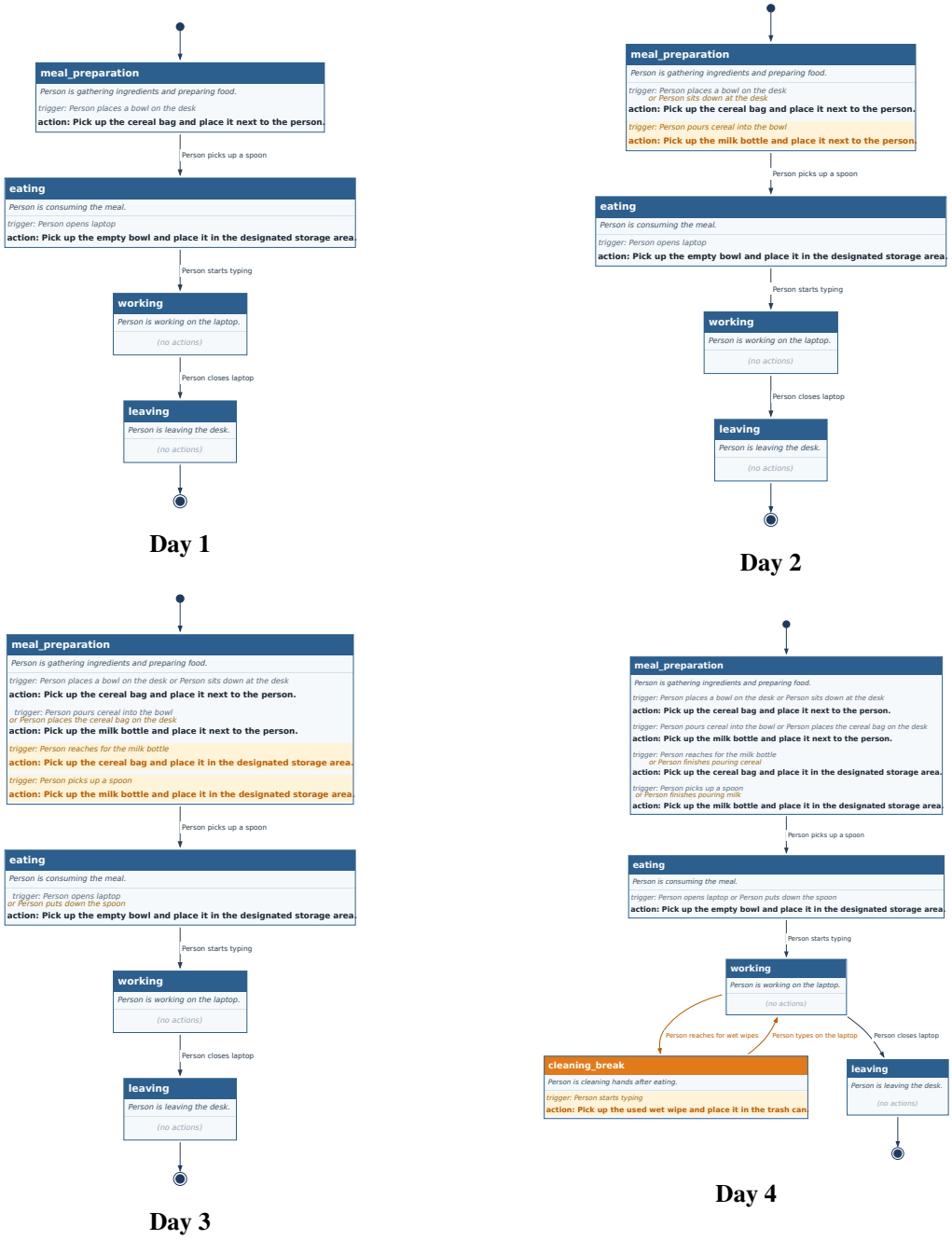

Figure 12: FSM memory after each of the four end-of-day reflections (one chain run). New states, `(trigger, action)` pairs, and transitions are highlighted in orange; on existing pairs, only the newly added *or*-clauses inside a trigger are orange, while original clauses stay gray.

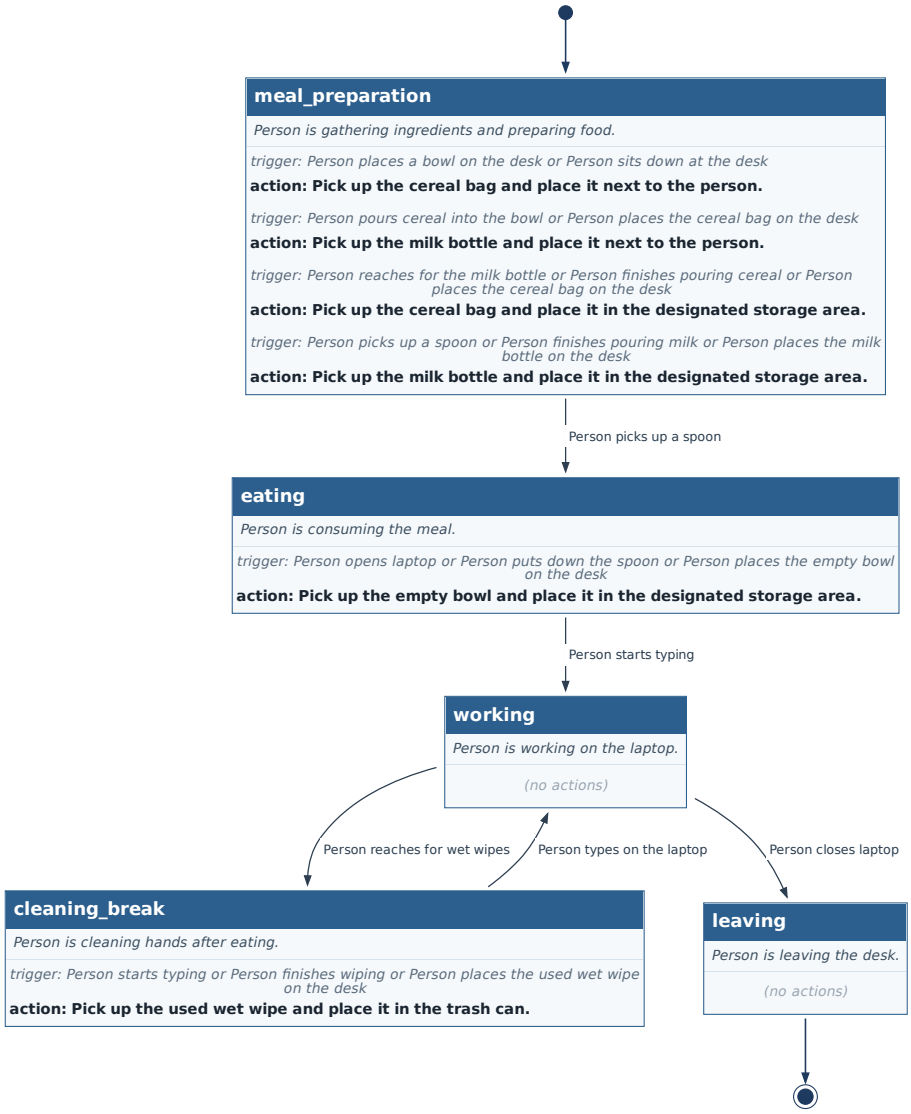

Figure 13: Our final FSM-structured memory instance after the five evaluation days. States are activity phases of the user's routine; per-state rules list `(trigger, action)` pairs that fire while the FSM occupies that state; edges are event-driven transitions between states.

C.2 Reasoning Trace of Memory Update

As shown in our overview freamework (Figure 2(b) of the main paper), at the end of each day, the reflector reads the day's video together with the current FSM memory and proposes updates. To illustrate how our VLM reasons about these updates, we walk through three excerpts of the FSM reflector's trace that exemplify three kinds of update: (*i*) the introduction of a new `(trigger, action)` pair, (*ii*) the extension of an existing trigger to fire earlier, and (*iii*) the introduction of a new state.

Introducing a new `(trigger, action)` pair. Figure 14 shows the reflector building the initial memory from the first visited day's video. After laying out the meal-preparation rules, the reflector wants to add a rule for putting the cereal bag back into storage but realizes it cannot trigger on the

```
...

|I cannot trigger "Pick up cereal bag" based on the person handling the cereal bag.|
I must find a different cue.
Maybe "Person finishes pouring cereal"?
At Frame 41-48, person is pouring/holding bag.
If trigger is "Person finishes pouring cereal", person is holding bag. Forbidden.

Is there any other cue?
|What if the trigger is "Person reaches for the milk bottle"?
If "Person reaches for the milk bottle", the person is touching the milk bottle. The
 robot touches the cereal bag. Different objects. This is VALID.|
Even if the human *already* moved the bag in the video, the rule is for future days.
 If the robot sees the person reach for milk, it assumes the cereal bag is nearby (
on the desk) and puts it away.
So, Trigger: "Person reaches for the milk bottle". Action: "Pick up the cereal bag
and place it in the designated storage area."
This seems plausible. |The logic is: "You are getting the next ingredient (milk), so
 I will tidy up the previous one (cereal)."|

...
```

Figure 14: Reflector trace excerpt from the initial FSM build. The reflector adds a put-cereal-back `(trigger, action)` pair to the `meal_preparation` state. Because the same-object constraint forbids triggering "pick up cereal bag" on the person handling the cereal bag itself, the reflector searches for an indirect cue and reasons explicitly about cross-object co-occurrence; the highlighted lines show the moment of decision and the underlying intent.

cereal bag itself (same-object constraint). Instead of giving up on the action, the reflector searches for an indirect cue and lands on "Person reaches for the milk bottle": the person is now in motion with the milk, so the robot is free to handle the cereal bag. The trace then verbalizes the underlying intent in plain English ("You are getting the next ingredient (milk), so I will tidy up the previous one (cereal)"), and writes the new pair into the `meal_preparation` state.

Extending an existing trigger. Figure 15 shows the same memory three days later, after the reflector has watched another video in which the human put the cereal bag away themselves. The cereal-storage `(trigger, action)` pair already exists in the `meal_preparation` state, but its trigger ("Person reaches for the milk bottle *or* Person places the cereal bag on the desk") did not fire because in this video the human walked the bag directly into storage without first setting it on the desk. The reflector inspects the video timeline, identifies that "finishes pouring cereal" happens just before the storage handoff, verifies the cross-object validity, and appends the new cue with *or*. The trigger is extended; no new pair is created.

Introducing a new state. The reflector occasionally restructures the FSM topology itself. Figure 16 shows the reflector debating whether to keep wet-wipe disposal inside the `working` state or to introduce a separate phase. It first considers folding the wiping action into `working` but rejects this because the routine instruction asks for a new state when the video reveals a distinct phase. It then sketches the bidirectional structure `working` $\rightarrow$ `cleaning_break` $\rightarrow$ `working`, briefly worries about state-ID uniqueness in the FSM schema, and settles on a single `working` state with a `cleaning_break` loop. The resulting layout bounds the wet-wipe `(trigger, action)` pair to the brief cleaning sub-phase, so the pair cannot fire on every typing cue, only after the `working` $\rightarrow$ `cleaning_break` transition has activated it.

Across these three update types (adding new pairs, broadening existing triggers, and introducing new states) the FSM container constrains each edit to a specific phase. The trigger broadening in Figure 15 stays inside `meal_preparation`, so the new cue cannot fire later in the day; the new state in the third example bounds the wet-wipe pair to the brief cleaning sub-phase. This phase-

```
...

Current Memory Rule: trigger: Person reaches for the milk bottle or Person places
the cereal bag on the desk
                   -> Action: Pick up the cereal bag and place it in the designated
storage area.
This rule exists. And the human performed the action (putting cereal away).
So I must extend the trigger for this rule.
Current trigger: Person reaches for the milk bottle or Person places the cereal bag
on the desk.
The human placed the cereal bag directly in storage (Step 4).
So the trigger "Person places the cereal bag on the desk" did not fire.
|Looking at the video:
  00:36 Person finishes pouring.
  00:38 Person stands bag up.
  00:40 Person moves bag to storage.
So `Person finishes pouring cereal` is a good earlier cue.|
|New trigger:| Person reaches for the milk bottle or Person places the cereal bag on
 the desk |or Person finishes pouring cereal.|

...
```

Figure 15: Reflector trace excerpt from a later day, showing trigger extension on the existing cereal-storage pair. The previous trigger missed the moment because the human walked the bag directly into storage without setting it on the desk. The highlighted lines show the reflector inspecting the video timeline to find an earlier, more reliable cue, and committing to "Person finishes pouring cereal" as the new cue to append with *or*.

local editing is what gives the FSM reflector its routine-aware behavior, and is the same scoping mechanism whose absence we analyzed in Section C.3.

C.3 Comparison of Memory Structures

The main paper showed that the FSM-structured PAM consistently outperforms List Memory on T1–T6. To expose the structural reason, we compare the two memories the reflector produces from the same single-user trace under otherwise identical settings, so any difference comes from the representation alone. We visualize the FSM instance as a graph (Fig. 13; YAML in Fig. 17) and the matching List instance as a flat rule list with no states or transitions (Fig. 18).

Why the FSM structure helps. The FSM memory does more than store `(trigger, action)` pairs: it also encodes the user's routine itself, as a graph of activity phases connected by transitions (Figure 13). At inference the predictor first locates which phase the user is currently in, and the transitions tell it where the routine will go next. The state graph thus supplies an additional layer of context that `(trigger, action)` pairs alone cannot, and the FSM is, in effect, a compact representation of how this particular user typically progresses through the day. The List Memory has no such representation; whatever routine knowledge it carries must be wedged into the trigger strings themselves, and the predictor has to recover the routine fresh from every new observation.

Figure 19: A task-predictor input frame during the eating phase: the user eats cereal while the milk and cereal are already in storage. Here the FSM predictor outputs `NO_ACT` while the list predictor fires T2 (Milk → Person).

An example where the structure pays off. An illustrative example of this appears during the eating phase (Figure 19). Throughout the meal the predictor sees the

```
...

Is it necessary?
The summary distinguishes "cleaning up" (wiping hands).
But functionally, it happens during the `working` phase (between typing sessions).
If I merge it into `working`, the description "Person is working on the laptop"
covers it loosely.
|However, the prompt says "Add a new state when the video reveals a distinct phase
of the routine that is not yet modeled."
Is wiping hands a distinct phase?
It interrupts the typing.|
Maybe `working` -> `cleaning_up` -> `working`?
Transitions:
  `working` -> `cleaning_up` on "Person reaches for wet wipes".
  `cleaning_up` -> `working` on "Person starts typing".
Rule in `cleaning_up`: Trigger "Person starts typing". Action "Pick up used wet wipe
 and place it in the trash can.".

...

I can call the second one `working_resumed`? Or just keep `working` and loop back?
FSM structure in JSON: `states` is a list of objects. IDs must be unique strings.
|So I need unique IDs. Or just `working` and `cleaning_break`.
Transition `working` -> `cleaning_break` on "Person reaches for wet wipes".
Transition `cleaning_break` -> `working` on "Person starts typing".|
Rule in `cleaning_break`: Trigger "Person starts typing". Action "Pick up the used
wet wipe and place it in the trash can."

...
```

Figure 16: Reflector trace excerpt from the initial FSM build, showing the decision to introduce a new `cleaning_break` state instead of folding wet-wipe disposal into `working`. The first highlighted block is the architectural justification ("the prompt says . . . a distinct phase of the routine"); the second highlighted block is the loop construction that bounds the wet-wipe pair to the cleaning sub-phase.

user sitting at the desk eating cereal with a spoon, with the cereal bag and milk bottle already returned to the storage area earlier in the routine. Both predictors describe the scene correctly as "person holding a bowl and a spoon, eating cereal, with milk and cereal already in storage." The FSM predictor, reading the memory shown in Figures 13 and 17, locates the user in the `eating` state, which contains no Milk → Person action, and outputs `NO_ACT`. The predictor reading the List Memory in Figure 18 instead matches the cue "Person holds the bowl" to the Milk → Person action, whose trigger has been broadened during reflection to "Person pours cereal *or* finishes pouring cereal *or* holds the bowl." It then outputs T2 (Milk → Person) repeatedly, even though the human has already poured the milk and is now eating. This failure is because whether the cue "holds the bowl" should trigger an action depends on the user's current phase: it legitimately fires Milk → Person at meal preparation, but should not trigger any action during eating.

How the over-broadened trigger entered the memory. The reflector's own thinking trace makes the origin of this failure explicit. After processing the previous day, the list reflector observed that the human had picked up the milk bottle without robot assistance and concluded that the trigger for the Milk → Person action must have been too rigid; it then brainstormed additional cues to append with *or*. The only constraint it applied was syntactic ("trigger object (bowl) ≠ action object (milk), valid"), and "Person holds the bowl" passed this check and was appended verbatim into the trigger for the Milk → Person action. The reflector never evaluated whether "holds the bowl" is temporally specific to meal preparation; it had no way to, because the List Memory has no notion of phase, and trigger specificity in the List representation must be encoded in the natural-language string itself rather than by where the pair sits. Even if the FSM reflector had nominated the same cue, the state container would have placed it inside the `meal_preparation` state, and the predictor would never

```
initial_state: meal_preparation
states:
- id: meal_preparation
  description: Person is gathering ingredients and preparing food.
  rules:
  - trigger: Person places a bowl on the desk or Person sits down at the desk
    action: Pick up the cereal bag and place it next to the person.
  - trigger: Person pours cereal into the bowl or Person places the cereal bag on
      the desk
    action: Pick up the milk bottle and place it next to the person.
  - trigger: Person reaches for the milk bottle or Person finishes pouring cereal
      or Person places the cereal bag on the desk
    action: Pick up the cereal bag and place it in the designated storage area.
  - trigger: Person picks up a spoon or Person finishes pouring milk or Person
places
      the milk bottle on the desk
    action: Pick up the milk bottle and place it in the designated storage area.
  next:
  - to: eating
    'on': Person picks up a spoon
- id: eating
  description: Person is consuming the meal.
  rules:
  - trigger: Person opens laptop or Person puts down the spoon or Person places the
      empty bowl on the desk
    action: Pick up the empty bowl and place it in the designated storage area.
  next:
  - to: working
    'on': Person starts typing
- id: working
  description: Person is working on the laptop.
  rules: []
  next:
  - to: cleaning_break
    'on': Person reaches for wet wipes
  - to: leaving
    'on': Person closes laptop
- id: cleaning_break
  description: Person is cleaning hands after eating.
  rules:
  - trigger: Person starts typing or Person finishes wiping or Person places the
used
      wet wipe on the desk
    action: Pick up the used wet wipe and place it in the trash can.
  next:
  - to: working
    'on': Person types on the laptop
- id: leaving
  description: Person is leaving the desk.
  rules: []
  next: []
```

Figure 17: Underlying YAML for the *Ours (FSM Memory)* instance in Figure 13.

have considered it during eating. The phase structure thus helps at two points, not one: it gives the predictor extra context at inference, and it equally gives the reflector a place to scope each trigger so that trigger-broadening cannot accidentally over-fire later phases. In short, the same syntactic-only check that broke in the List case would have produced an identical cue under FSM, but the cue's placement inside `meal_preparation` would have rendered it harmless during eating.

```
rules:
- trigger: Person sits at the desk or Person approaches the desk or Person enters
    the room or Person stands at the desk or Person walks towards the desk
  action: Pick up the bowl and place it next to the person.
- trigger: Person pours cereal into the bowl or Person finishes pouring cereal or
    Person holds the bowl or Person places the cereal bag on the desk
  action: Pick up the milk bottle and place it next to the person.
- trigger: Person places the bowl on the desk or Person sits at the desk or Person
    holds the bowl
  action: Pick up the cereal bag and place it next to the person.
- trigger: Person types on the laptop or Person rests hands on the keyboard or
Person
    finishes eating
  action: Pick up the wet wipe package and place it next to the person.
- trigger: Person pours milk into the bowl
  action: Pick up the cereal bag and place it in the designated storage area.
- trigger: Person eats from the bowl
  action: Pick up the milk bottle and place it in the designated storage area.
- trigger: Person closes the laptop
  action: Pick up the bowl and place it in the designated storage area.
```

Figure 18: Underlying YAML for the matching *Ours (List Memory)* instance after the five evaluation days: a list of `(trigger, action)` pairs with no state grouping or transitions.

D Diverse Scenario Analysis

In the following sections, we present two additional experiments under varying scenarios, addressing two research questions: (*i*) whether our memory generalizes to other scenarios (Section D.1), and (*ii*) whether a memory already built on one scenario can adapt to a different one (Section D.2).

D.1 Generalization to a Different Routine

Figure 20: **A different routine (medication scenario).** The person enters empty-handed, takes a pill with water, then drinks a can of Coke while working on the laptop before leaving. The robot handles six atomic tasks (T1: deliver the medicine, T2: deliver the water, T3: return the medicine bottle, T4: return the water bottle, T5: deliver the Coke, T6: discard the empty Coke can) in a routine whose intent cues are far weaker than the main scenario.

A More Challenging Routine with Weaker Intent Cues. To test whether our memory generalizes beyond the cereal routine, we design a second, structurally different routine: a short medication break. The person enters the desk empty-handed, takes a pill with water, and then drinks a can of Coke while working on the laptop before leaving. The robot is responsible for six atomic tasks: (T1) deliver the medicine, (T2) deliver the water, (T3) return the medicine bottle to storage, (T4) return the water bottle to storage, (T5) deliver the Coke, and (T6) discard the empty Coke can. Compared with the main scenario, this routine is substantially harder for a proactive predictor: the user's intent is far less legible from the visible scene alone. In the cereal routine, the act of pouring cereal strongly signals that milk should follow; here, by contrast, entering empty-handed gives no object-level cue about which item is needed next, and the pairing between medicine, water, and Coke is a matter of personal habit rather than physical necessity. Reading the correct action therefore requires the accu-

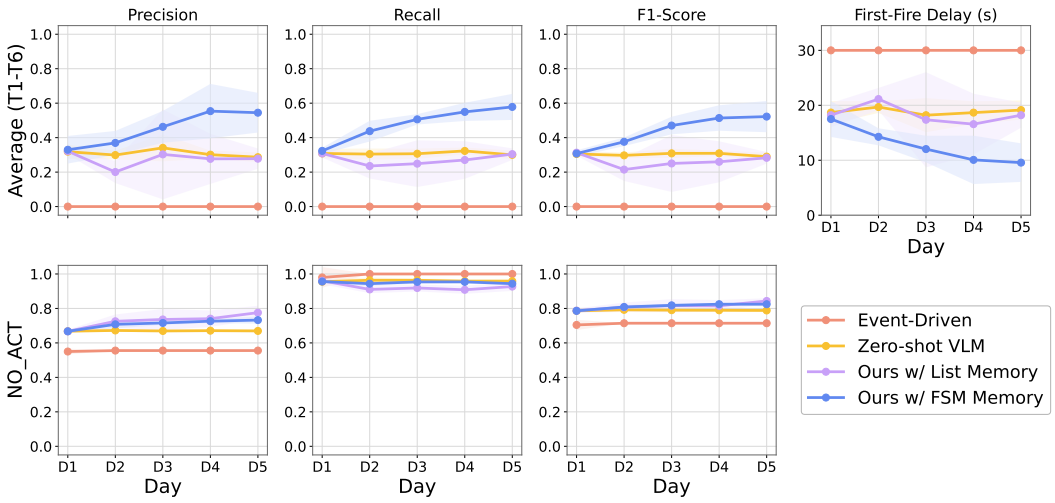

Figure 21: **Results on the medication scenario across five days.** Per-day precision, recall, and F1 averaged over the six proactive tasks (top) and on `NO_ACT` (bottom), with first-fire delay (top right). Even in this harder, low-cue routine, Ours w/ FSM Memory improves day over day and leads on precision, recall, and F1, reaching the shortest first-fire delay by Day 5; the memory-free Event-Driven baseline fails to fire at all on T1–T6.

mulated routine knowledge that memory is meant to provide, making this a stress test of whether a method can act proactively when the scene itself is uninformative.

Our Memory Generalizes Beyond a Single Routine. Our FSM memory transfers cleanly to this harder routine. On the six proactive tasks it improves steadily from Day 1 to Day 5 in precision, recall, and F1, and reduces its first-fire delay from roughly 17 s to about 10 s, indicating that the reflector accumulates and sharpens triggers here just as in the main scenario. List Memory lags well behind and stays largely flat across days, showing that the structural advantage of the FSM persists when intent cues are weak: without phase grouping, the list predictor cannot reliably tell which item the user needs at each point of an already ambiguous routine.

Weak Cues Break Memory-Free Baselines. The difficulty of this routine is most visible in the Event-Driven baseline, which sits near zero on every proactive metric across all five days. Having no memory and reacting only to the immediate scene, it finds no signal to react to in this low-cue setting and defaults to inaction; this conservatism keeps its `NO_ACT` score high but leaves it unable to help with any of T1–T6. The Zero-shot VLM does fire, but without a mechanism to retain experience it neither improves over the five days nor shortens its high, flat first-fire delay. Together these results show that our gains are not specific to a single routine: even when the user's intent is barely readable from the scene, the self-evolving memory still converts repeated exposure into faster, more accurate assistance, in a setting where a memory-free reactive policy fails outright.

D.2 Memory Adaptation to Routine Changes

We refer to the main paper's routine (cereal / milk / wet-wipe) as **Scenario A**, and the routine in Fig. 20 (medicine / water / Coke) as **Scenario B**. So far each scenario was built in isolation from an empty start. Here we ask what happens when a user who has lived with the system under one routine switches to another. We are not asking whether the old-routine memory helps on the new routine. We are asking the opposite two questions: *(i)* does leaving the old-routine memory in place actively hurt performance on the new one (because the wrong vocabulary fires false positives), and *(ii)* if the reflector keeps updating, how quickly can it shed the obsolete pieces and pick up the new structure?

Setup. We test both directions, A→B and B→A, comparing four conditions on the target routine's five-day stream with $n = 10$ seeds. The label scheme is `Memory(`⟨`init`⟩`)-`⟨`update mode`⟩`:` ⟨`init`⟩

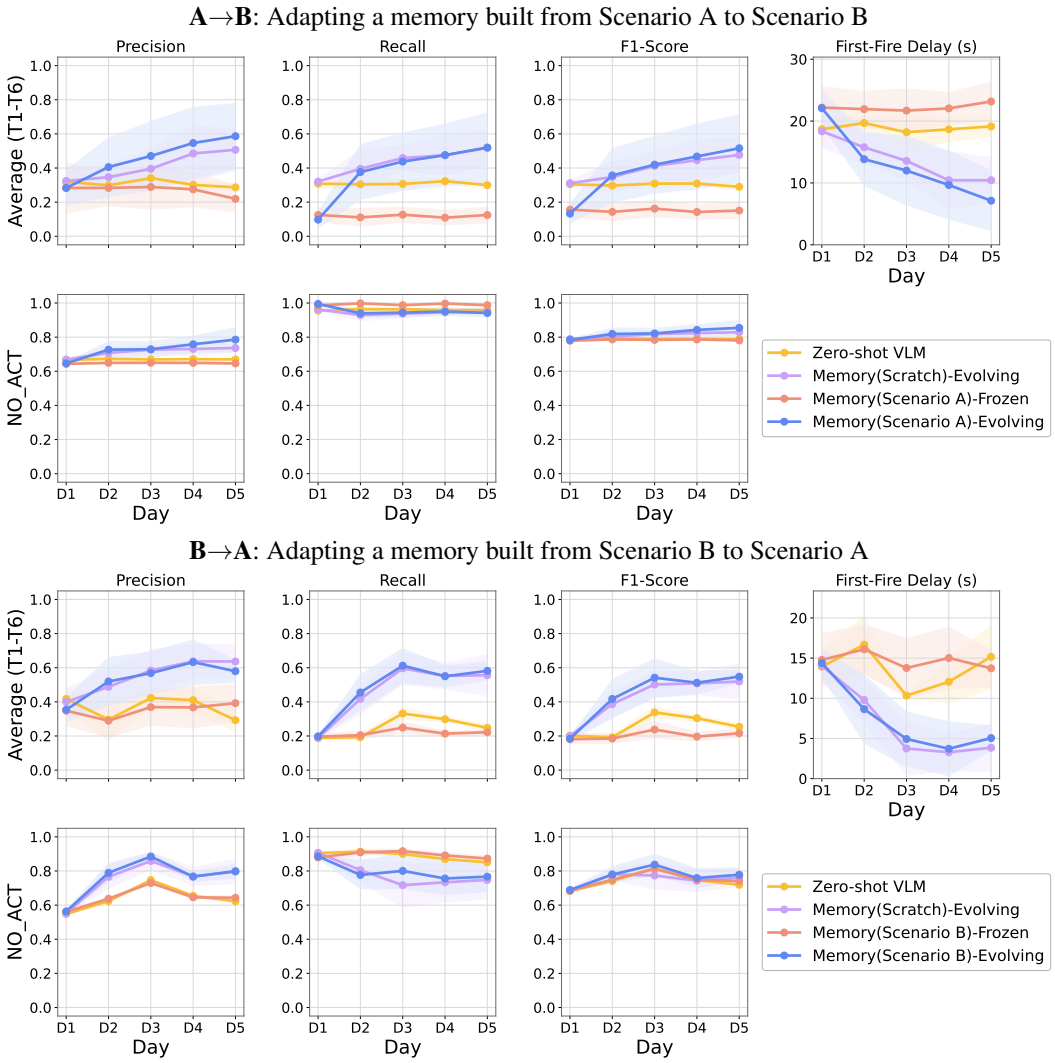

Figure 22: Memory adaptation across scenarios (A→B, top; B→A, bottom). In both directions, the evolving condition is slightly behind the from-scratch baseline on Day 1 but adapts within one or two days to match it, while the frozen source-memory underperforms throughout.

is what the memory starts as on Day 1 (an empty stub, written `Scratch`, or the source routine's Day-5 memory, written `Scenario A` or `Scenario B`), and ⟨`update mode`⟩ is whether the reflector keeps updating after each day (`Evolving`) or reuses the same memory unchanged across all five target days (`Frozen`).

- **Zero-shot VLM**: no memory at all; the predictor sees only the current frame window.
- **Memory(Scratch)-Evolving**: the standard self-evolving FSM memory of the main paper, started from an empty stub on the target routine and refined after each day. Cold-start baseline.
- **Memory(Scenario A)-Frozen** (A→B) / **Memory(Scenario B)-Frozen** (B→A): the source routine's Day-5 memory is loaded once and used statically across all five target days, with no reflection. This isolates the cost of leaving a stale memory in place.
- **Memory(Scenario A)-Evolving** (A→B) / **Memory(Scenario B)-Evolving** (B→A): the same source-routine Day-5 memory is loaded as the starting state, but the reflector keeps refining it after every target day. This condition measures how quickly the existing memory adapts to the new routine.

Seeds are paired across conditions: the source memory used in the `Evolving` and `Frozen` conditions for run i is the Day-5 memory produced by run i of the same routine's main experiment. The target-side five-day stream is identical to the corresponding scenario's main experiment, so the `Memory(Scratch)-Evolving` curve can be compared directly against earlier figures.

A stale memory hurts, but only if frozen. The Frozen condition consistently falls below the Zero-shot VLM on precision, recall, F1, and first-fire delay in both directions, confirming that an old-routine memory left unupdated is a liability when the routine changes: its old-routine vocabulary fires false positives on the new content. This is the failure mode one might worry about when reusing memory across routines, but it appears only when reflection is switched off.

With continued reflection, the existing memory adapts within a day or two. Once the reflector keeps updating, starting from the wrong routine costs nothing lasting. In both A$\rightarrow$B and B$\rightarrow$A, the Evolving condition is slightly behind the from-scratch baseline on Day 1 but matches it by Day 3, and from then on continues to refine the new triggers exactly as it would from an empty start. The mechanism is direct: within the first one or two days the reflector replaces obsolete source-routine pairs (e.g., "cereal $\rightarrow$ person") with target-routine pairs (e.g., "medicine $\rightarrow$ person"). A deployed system can therefore keep its memory through routine changes without the operator having to wipe it; the same reflector that built the original memory reshapes it for the new routine.

E Task Predictor Prompts

We provide the system prompts used by the inference VLM to decide, at each tick, whether to issue a task (and which one) given the most recent W frames plus the current memory. We include three prompts: the no-memory baseline (memory section absent), and one per memory variant (*Ours (List Memory)* and *Ours (FSM Memory)*). At inference time the user-turn carries the current wall-clock time and the most recent $W{=}5$ frames; the system prompts are reproduced verbatim below.

E.1 No Memory (baseline)

```
You are a task inference module for a household robot that supports everyday home
life.

# Environment
- The setting is a personal desk where a single person performs everyday activities.
- The designated storage area: the corner of the desk, where items such as cereal,
bowls, milk, drinks, and water are normally kept.

# Proactive Assistance Goals
- bringing the person an object they may need next.
- putting away objects the person appears to have finished using.
- etc. -- any helpful pick-and-place action that is grounded in observed behavior.

## Inputs
- A sequence of camera frames, ordered from oldest to newest and sampled at 1 fps (
up to {window} frames), together with the corresponding wall-clock time of the final
 frame. Each frame is the horizontal concatenation of two back-facing cameras (
left_back on the left, right_back on the right).

## Objective
- Given the frames, select exactly one task by index only if it is appropriate and
can be executed immediately at the current moment based on the final (current) frame
.
- As long as the selected task is appropriate to the current situation, executing it
 does not interfere with the human. If no task should be executed at the current
moment, respond with `task_index = -1`.
- If the person is holding or touching an object, the robot cannot perform any
action on that object until the person sets it down on the desk.

## Available Tasks
{tasks_block}

## Output Requirements
- `observation` should be a single concise sentence describing the current scene
state, focusing on task-relevant objects, humans, and state changes.
- `reasoning` should be a single concise sentence on whether and why a task should
fire now.
- `task_index` should be either `-1` for no action or the index of a task from
Available Tasks.

## Output Schema
Respond with a single JSON object and nothing else (no prose, no markdown fences):
{
  "observation": "<single concise sentence describing the relevant current scene
state>",
  "reasoning":   "<single concise sentence on whether an immediate robot action
would help the human right now>",
  "task_index":  <integer: -1 for no action, otherwise 0..{n_tasks_minus_one}>
}
```

E.2 Ours (List Memory)

```
You are a task inference module for a household robot that supports everyday home
life.

# Environment
- The setting is a personal desk where a single person performs everyday activities.
- The designated storage area: the corner of the desk, where items such as cereal,
bowls, milk, drinks, and water are normally kept.

# Proactive Assistance Goals
- bringing the person an object they may need next.
- putting away objects the person appears to have finished using.
- etc. -- any helpful pick-and-place action that is grounded in observed behavior.

## Inputs
- A sequence of camera frames, ordered from oldest to newest and sampled at 1 fps (
up to {window} frames), together with the corresponding wall-clock time of the final
 frame. Each frame is the horizontal concatenation of two back-facing cameras (
left_back on the left, right_back on the right).
- A procedural MEMORY built by reflecting on the user's past activities.

## Objective
- Given the frames and the memory, select exactly one task by index only if it is
appropriate and can be executed immediately at the current moment based on the final
 (current) frame.
- As long as the selected task is appropriate to the current situation, executing it
 does not interfere with the human. If no task should be executed at the current
moment, respond with `task_index = -1`.
- If the person is holding or touching an object, the robot cannot perform any
action on that object until the person sets it down on the desk.

## Memory
- The memory is a flat list of production rules; each rule = (NL trigger, NL action)
. The trigger is a single short natural-language statement describing the visible
event/moment that causes the rule to fire.
- Treat the memory as a REFERENCE, not as a strict matcher. Do not over-fixate on
exact trigger wording: if a memory action is semantically appropriate for the
current scene, you may fire it even when the trigger phrasing doesn't match the
scene verbatim.

{rules_block}

## Available Tasks
{tasks_block}

## Output Requirements
- `observation` should be a single concise sentence describing the current scene
state, focusing on task-relevant objects, humans, and state changes.
- `reasoning` should be a single concise sentence on whether and why a task should
fire now, referencing the relevant rule of the memory.
- `task_index` should be either `-1` for no action or the index of a task from
Available Tasks.

## Output Schema
Respond with a single JSON object and nothing else (no prose, no markdown fences):
{
  "observation": "<single concise sentence describing the relevant current scene
state>",
  "reasoning":   "<single concise sentence on whether an immediate robot action
would help the human right now, referencing the relevant rule of the memory>",
  "task_index":  <integer: -1 for no action, otherwise 0..{n_tasks_minus_one}>
}
```

E.3 Ours (FSM Memory)

```
You are a task inference module for a household robot that supports everyday home
life.

# Environment
- The setting is a personal desk where a single person performs everyday activities.
- The designated storage area: the corner of the desk, where items such as cereal,
bowls, milk, drinks, and water are normally kept.

# Proactive Assistance Goals
- bringing the person an object they may need next.
- putting away objects the person appears to have finished using.
- etc. -- any helpful pick-and-place action that is grounded in observed behavior.

## Inputs
- A sequence of camera frames, ordered from oldest to newest and sampled at 1 fps (
up to {window} frames), together with the corresponding wall-clock time of the final
 frame. Each frame is the horizontal concatenation of two back-facing cameras (
left_back on the left, right_back on the right).
- A procedural MEMORY built by reflecting on the user's past activities.

## Objective
- Given the frames and the memory, select exactly one task by index only if it is
appropriate and can be executed immediately at the current moment based on the final
 (current) frame.
- As long as the selected task is appropriate to the current situation, executing it
 does not interfere with the human. If no task should be executed at the current
moment, respond with `task_index = -1`.
- If the person is holding or touching an object, the robot cannot perform any
action on that object until the person sets it down on the desk.

## Memory
- The memory is structured as a hybrid finite-state machine: each state captures a
phase of the user's routine, inside each state a small list of production rules
describes how the robot should help while the user is in that phase, and event-
driven transitions describe how the routine progresses between phases. Each rule's
trigger is a single short natural-language statement describing the visible event/
moment that causes the rule to fire.
- Use the memory holistically as structured context -- you do not need to commit to
a single state; rather, use the state descriptions, per-state rules, and transitions
 together to judge whether any rule from the memory fires for the current scene.
- Treat the memory as a REFERENCE, not as a strict matcher. Do not over-fixate on
exact trigger wording or current state: if a memory action is semantically
appropriate for the current scene, you may fire it even when the trigger phrasing or
 its state assignment doesn't match the scene verbatim.

{fsm_block}

## Available Tasks
{tasks_block}

## Output Requirements
- `observation` should be a single concise sentence describing the current scene
state, focusing on task-relevant objects, humans, and state changes.
- `reasoning` should be a single concise sentence on whether and why a task should
fire now, referencing the relevant phase or rule of the memory.
- `task_index` should be either `-1` for no action or the index of a task from
Available Tasks.

## Output Schema
Respond with a single JSON object and nothing else (no prose, no markdown fences):
{
  "observation": "<single concise sentence describing the relevant current scene
state>",
```

```
  "reasoning":   "<single concise sentence on whether an immediate robot action
would help the human right now, referencing the relevant phase or rule of the memory
>",
  "task_index":  <integer: -1 for no action, otherwise 0..{n_tasks_minus_one}>
}
```

F Reflector Prompts

We provide the system prompts used by the reflection VLM to either *build* the procedural memory from scratch (no prior memory) or *update* the memory at end-of-day given a prior memory and the day's observations. Each prompt below corresponds to one of the two memory variants we ablate: *Ours (List Memory)* (a flat list of production rules) and *Ours (FSM Memory)* (a hybrid finite-state machine with state-conditional rules and event-driven transitions). The user-turn message simply announces the day index plus the day's video frames; the system prompts are reproduced verbatim below.

F.1 Ours (List Memory): build (no prior memory)

```
You are observing a person's daily activities to build a procedural memory of when
and how a household robot should help them.

# Environment
- The setting is a personal desk where a single person performs everyday activities.
- The designated storage area: the corner of the desk, where items such as cereal,
bowls, milk, drinks, and water are normally kept.

# Proactive Assistance Goals
- bringing the person an object they may need next.
- putting away objects the person appears to have finished using.
- etc. -- any helpful pick-and-place action that is grounded in observed behavior.

# Inputs
- The training video: the person's daily routine recorded from two back-facing
cameras concatenated side by side (left_back (*@\textbar@*) right_back). This is the
 very first day -- there is no prior memory, no summary, and no human feedback yet.

# Task
- Given the training video, BUILD a (trigger, action) rule memory FROM SCRATCH.
- The memory will be used by the robot to proactively assist the person in future
days, so the goal is to produce a memory that helps the person as much as possible.

# Build Instructions
- Use the training video to build the memory rules.
- First, explicitly RE-STATE the pick-and-place actions that the human performed in
this video. Then, for EACH of those human actions, think concretely about HOW the
robot could have taken that action over on the person's behalf (e.g., by bringing
the object to the person earlier, or putting it away afterwards). Your rules should
be built around these take-over opportunities -- do NOT invent rules for objects or
actions the human did not interact with in this video.
- Each rule's action MUST literally mirror an action the human performed in this
video: the picked-up object, the destination, and the object→destination motion
must all correspond to something the human actually did. Do NOT invent hypothetical
actions.
- If a trigger describes the person already in motion with an object (e.g., "reaches
 for X", "picks up X", "pours X"), the object the person is in motion with MUST be
DIFFERENT from the object the rule's action manipulates. Same-object triggers are
forbidden -- e.g., trigger "Person reaches for the cereal bag" paired with action "
Pick up the cereal bag and place it next to the person" is invalid, because the
person is already fetching that same cereal bag themselves so the robot is too late.
 Cross-object triggers are fine -- e.g., trigger "Person pours cereal into the bowl"
 paired with action "Pick up the milk bottle and place it next to the person" is
```

```
valid, because the person is busy with the cereal while the robot preempts on the
milk.
- Deployment caveat: when this memory is deployed, the human will act expecting the
robot to help. So any rule whose trigger is phrased as "person picks up X", "person
fetches X", or "person moves X" risks never firing -- the robot would have picked X
up first, leaving no human action to observe. Each rule's trigger must describe an
observable cue the human will produce REGARDLESS of robot assistance (e.g. sitting
down at the desk, placing an item they have just finished using, finishing or
discarding a consumable, opening a wrapper they must open themselves). Avoid
triggers that depend on the human performing an action the robot might preempt.
- Identify EVERY moment in the video where the robot could helpfully act and write
one rule per such moment, using only observable cues -- the routine typically holds
multiple rules unless the moment is purely receptive (e.g. the human is actively
eating).
- Maximize action coverage -- the deployed robot's only purpose is to take over
human actions, so MORE rules covering every take-over moment is strictly better than
 fewer, AS LONG AS each rule mirrors a real human-performed action from this video.
Conversely, an action the human never performed in this video is OUT OF SCOPE --
never invent a rule for it, no matter how plausible it might seem for a household
robot in general.
- If the person is holding or touching an object, the robot cannot perform any
action on that object until the person sets it down on the desk.

# Output explanations
- trigger: a single short natural-language statement describing the visible event/
moment that should cause this rule's action to fire. The trigger MUST be self-
contained -- a proactive robot, reading ONLY the trigger (without seeing prior
memory, state assignment, or surrounding context), should find the corresponding
action to be a clearly sensible, obvious response.
- action: a NATURAL LANGUAGE description of how the robot should help, written in
the canonical form `Pick up <object> and place it <location>.` Valid `<location>`
values are exactly one of: `next to the person`, `in the designated storage area`, `
in the trash can`.

# Output (JSON only, no markdown, no commentary)
{
  "rules": [
    {"trigger": "<NL statement>", "action": "<string>"},
    ...
  ]
}
```

F.2 Ours (List Memory): end-of-day update

```
You are observing a person's daily activities to build a procedural memory of when
and how a household robot should help them.

# Environment
- The setting is a personal desk where a single person performs everyday activities.
- The designated storage area: the corner of the desk, where items such as cereal,
bowls, milk, drinks, and water are normally kept.

# Proactive Assistance Goals
- bringing the person an object they may need next.
- putting away objects the person appears to have finished using.
- etc. -- any helpful pick-and-place action that is grounded in observed behavior.

# Inputs
- Current memory: the (trigger, action) entries that were applied today
- Today's video: the person's activities recorded from two back-facing cameras
concatenated side by side (left_back (*@\textbar@*) right_back)
- Summary: what the person and robot each did in today's video, in temporal order
- Human feedback: today's feedback from the person, may be absent
```

```
# Task
- Given the current memory, today's video, the summary, and human feedback, update
the memory of (trigger, action) rules.
- The memory will be used by the robot to proactively assist the person in future
days, so the goal is to produce a memory that helps the person as much as possible.

# Update Instructions
- Use the video, summary, action history, current memory, and human feedback to
update the memory rules.
- First, explicitly RE-STATE the pick-and-place actions that the human performed in
this video and summary. Then, for EACH of those human actions, think concretely
about HOW the robot could have taken that action over on the person's behalf (e.g.,
by bringing the object to the person earlier, or putting it away afterwards). Your
rules should be built around these take-over opportunities -- do NOT invent rules
for objects or actions the human did not interact with in this video and summary.
- Each rule's action MUST literally mirror an action the human performed in this
video and summary: the picked-up object, the destination, and the object→
destination motion must all correspond to something the human actually did. Do NOT
invent hypothetical actions.
- If a trigger describes the person already in motion with an object (e.g., "reaches
 for X", "picks up X", "pours X"), the object the person is in motion with MUST be
DIFFERENT from the object the rule's action manipulates. Same-object triggers are
forbidden.
- Deployment caveat: when this memory is deployed, the human will act expecting the
robot to help. So any rule whose trigger is phrased as "person picks up X", "person
fetches X", or "person moves X" risks never firing -- the robot would have picked X
up first, leaving no human action to observe. Each rule's trigger must describe an
observable cue the human will produce REGARDLESS of robot assistance.
- Default to ADDING a new rule whenever the summary mentions a human pick-and-place
action that the robot could plausibly take over and that is not yet covered by the
current memory. It is better to add a rule that may need refinement later than to
omit a useful take-over opportunity. Err on the side of MORE rules, not fewer.
- Refine an existing rule when its trigger or action is too rigid, too loose,
ambiguous, or otherwise misaligned with helpful robot behavior.
- IF AND ONLY IF a memory rule's action is in the memory AND the summary shows the
human (not the robot) performed that action: this is most likely NOT because the
action is unnecessary, but because the rule's trigger was too strict, so the robot
did not fire -- and consequently the person ended up doing it themselves. For ONLY
that specific rule, extend its trigger by appending an additional observable cue
with `or` (e.g., `<original cue> or <a cue that occurs earlier or is more visually
distinct>`). Keep the original cue intact and just append. Do NOT modify the
triggers of any other rules -- only the rule whose action was performed by the human
.
- Remove a rule only when it is clearly wrong, such as when the trigger fired but
the action would have been inappropriate.
- Before finalizing the updated memory, mentally re-apply it to TODAY'S video as if
you were the inference module: walk through each video moment and ask "would this
trigger fire when intended? would it mis-fire when it shouldn't?" If a rule would
mis-fire or fail to fire at its intended moment, revise its trigger (or remove the
rule) before producing the final output. Use the thinking trace to make this
verification pass explicit.
- Do not remove a rule simply because its trigger did not occur in today's video.
- If human feedback is provided, treat it as authoritative and prioritize it over
your own judgment.

# Output explanations
- trigger: a single short natural-language statement describing the visible event/
moment that should cause this rule's action to fire. The trigger MUST be self-
contained.
- action: a NATURAL LANGUAGE description of how the robot should help, written in
the canonical form `Pick up <object> and place it <location>.` Valid `<location>`
values are exactly one of: `next to the person`, `in the designated storage area`, `
in the trash can`.
```

```
# Output (JSON only, no markdown, no commentary)
{
  "rules": [
    {"trigger": "<NL statement>", "action": "<string>"},
    ...
  ]
}
```

F.3 Ours (FSM Memory): build (no prior memory)

```
You are observing a person's daily activities to build a procedural memory of when
and how a household robot should help them.

# Environment
- The setting is a personal desk where a single person performs everyday activities.
- The designated storage area: the corner of the desk, where items such as cereal,
bowls, milk, drinks, and water are normally kept.

# Proactive Assistance Goals
- bringing the person an object they may need next.
- putting away objects the person appears to have finished using.
- etc. -- any helpful pick-and-place action that is grounded in observed behavior.

# Inputs
- The training video: the person's daily routine recorded from two back-facing
cameras concatenated side by side (left_back (*@\textbar@*) right_back). This is the
 very first day -- there is no prior memory, no summary, and no human feedback yet.

# Task
- Given the training video, BUILD the FSM memory FROM SCRATCH.
- The memory will be used by the robot to proactively assist the person in future
days, so the goal is to produce a memory that helps the person as much as possible.

# Memory Structure
The memory is structured as a HYBRID FINITE-STATE MACHINE.
- The person's activities are segmented into temporally ordered PHASES (FSM states).
 Each state should correspond to a meaningfully different activity, not a fleeting
micro-moment.
- Each state holds a set of PRODUCTION RULES that fire while the FSM occupies that
state. A rule = (NL trigger, NL action). A state may have zero rules if no helpful
action applies during that phase (e.g., the human is actively eating), or multiple
rules if several distinct visible cues each warrant a different proactive action --
the rule count per state is determined by the evidence, not capped at one.
- A (trigger, action) pair MAY appear in multiple states if the same proactive step
is appropriate in more than one phase. Do not artificially restrict each rule to a
single state when the action is reasonable in several phases -- duplicate the rule
across all states where it applies.
- Between states, EVENT-DRIVEN TRANSITIONS define progression. Each `next` entry: (
to: <state_id>, on: <NL event>). Transitions are not robot actions. Terminal states
use next: [].
- Exactly one state is the `initial_state`.

# Build Instructions
- Use the training video to build the FSM memory.
- First, explicitly RE-STATE the pick-and-place actions that the human performed in
this video. Then, for EACH of those human actions, think concretely about HOW the
robot could have taken that action over on the person's behalf (e.g., by bringing
the object to the person earlier, or putting it away afterwards). Your rules should
be built around these take-over opportunities -- do NOT invent rules for objects or
actions the human did not interact with in this video.
- Each rule's action MUST literally mirror an action the human performed in this
video: the picked-up object, the destination, and the object→destination motion
must all correspond to something the human actually did. Do NOT invent hypothetical
actions.
```

```
- If a trigger describes the person already in motion with an object (e.g., "reaches
 for X", "picks up X", "pours X"), the object the person is in motion with MUST be
DIFFERENT from the object the rule's action manipulates. Same-object triggers are
forbidden.
- Deployment caveat: when this memory is deployed, the human will act expecting the
robot to help. So any rule whose trigger is phrased as "person picks up X", "person
fetches X", or "person moves X" risks never firing -- the robot would have picked X
up first, leaving no human action to observe. Each rule's trigger must describe an
observable cue the human will produce REGARDLESS of robot assistance.
- Identify the natural temporal phases of the routine from the video and order them
as they appear.
- Within each phase, identify EVERY moment where the robot could helpfully act and
write one rule per such moment, using only observable cues -- a single phase
typically holds multiple rules unless the phase is purely receptive (e.g. the human
is actively eating).
- Define transitions on observable events that mark the end of one phase and the
start of the next.
- If the person is holding or touching an object, the robot cannot perform any
action on that object until the person sets it down on the desk.
- Pick exactly one phase as `initial_state` (the one that appears at the start of
the day), and ensure every `next.to` references an existing state id (terminal
phases use `next: []`).

# Output explanations
- trigger: a single short natural-language statement describing the visible event/
moment that should cause this rule's action to fire. The trigger MUST be self-
contained.
- action: a NATURAL LANGUAGE description of how the robot should help, written in
the canonical form `Pick up <object> and place it <location>.` Valid `<location>`
values are exactly one of: `next to the person`, `in the designated storage area`, `
in the trash can`.

# Output (JSON only, no markdown, no commentary)
{
  "initial_state": "<state_id>",
  "states": [
    {
      "id": "<snake_case>",
      "description": "<one-sentence>",
      "rules": [
        {"trigger": "<NL statement>", "action": "<NL>"},
        ...
      ],
      "next": [{"to": "<state_id>", "on": "<NL event>"}]
    }
  ]
}
```

F.4 Ours (FSM Memory): end-of-day update

```
You are observing a person's daily activities to build a procedural memory of when
and how a household robot should help them.

# Environment
- The setting is a personal desk where a single person performs everyday activities.
- The designated storage area: the corner of the desk, where items such as cereal,
bowls, milk, drinks, and water are normally kept.

# Proactive Assistance Goals
- bringing the person an object they may need next.
- putting away objects the person appears to have finished using.
- etc. -- any helpful pick-and-place action that is grounded in observed behavior.
```

```
# Inputs
- Current memory: the FSM (states, per-state (trigger, action) entries, and
transitions) that was applied today
- Today's video: the person's activities recorded from two back-facing cameras
concatenated side by side (left_back (*@\textbar@*) right_back)
- Summary: what the person and robot each did in today's video, in temporal order
- Human feedback: today's feedback from the person, may be absent

# Task
- Given the current memory, today's video, the summary, and human feedback, update
the FSM memory.
- The memory will be used by the robot to proactively assist the person in future
days, so the goal is to produce a memory that helps the person as much as possible.

# Memory Structure
The memory is structured as a HYBRID FINITE-STATE MACHINE.
- The person's activities are segmented into temporally ordered PHASES (FSM states).
 Each state should correspond to a meaningfully different activity, not a fleeting
micro-moment.
- Each state holds a set of PRODUCTION RULES that fire while the FSM occupies that
state. A rule = (NL trigger, NL action). A state may have zero rules if no helpful
action applies during that phase, or multiple rules if several distinct visible cues
 each warrant a different proactive action.
- A (trigger, action) pair MAY appear in multiple states if the same proactive step
is appropriate in more than one phase.
- Between states, EVENT-DRIVEN TRANSITIONS define progression. Each `next` entry: (
to: <state_id>, on: <NL event>). Transitions are not robot actions. Terminal states
use next: [].
- Exactly one state is the `initial_state`.

# Update Instructions
- Use the video, summary, current memory, and human feedback to update the FSM
memory.
- First, explicitly RE-STATE the pick-and-place actions that the human performed in
this video and summary. Then, for EACH of those human actions, think concretely
about HOW the robot could have taken that action over on the person's behalf. Your
rules should be built around these take-over opportunities -- do NOT invent rules
for objects or actions the human did not interact with.
- Each rule's action MUST literally mirror an action the human performed in this
video and summary. Do NOT invent hypothetical actions.
- If a trigger describes the person already in motion with an object, the object the
 person is in motion with MUST be DIFFERENT from the object the rule's action
manipulates. Same-object triggers are forbidden.
- Deployment caveat: when this memory is deployed, the human will act expecting the
robot to help. Each rule's trigger must describe an observable cue the human will
produce REGARDLESS of robot assistance.
- Add a new state when the video reveals a distinct phase of the routine that is not
 yet modeled.
- Default to ADDING a new rule whenever the summary mentions a human pick-and-place
action that the robot could plausibly take over and that is not yet covered by the
current memory. Err on the side of MORE rules, not fewer.
- Refine a state's description, rules, or transitions when they are too rigid, too
loose, ambiguous, or otherwise misaligned with helpful robot behavior.
- IF AND ONLY IF a memory rule's action is in the memory AND the summary shows the
human (not the robot) performed that action: extend its trigger by appending an
additional observable cue with `or`, and/or move or duplicate the rule to the
correct state. Keep the original cue intact and just append.
- Remove a state or rule only when it is clearly wrong, such as when the trigger
fired but the action would have been inappropriate.
- Do not remove a state or rule simply because its trigger did not occur in today's
video.
- Ensure `initial_state` names one of the states, and every `next.to` references an
existing state id.
- If human feedback is provided, treat it as authoritative and prioritize it over
your own judgment.
```

```
# Output explanations
- trigger: a single short natural-language statement describing the visible event/
moment that should cause this rule's action to fire. The trigger MUST be self-
contained.
- action: a NATURAL LANGUAGE description of how the robot should help, written in
the canonical form `Pick up <object> and place it <location>.` Valid `<location>`
values are exactly one of: `next to the person`, `in the designated storage area`, `
in the trash can`.

# Output (JSON only, no markdown, no commentary)
{
  "initial_state": "<state_id>",
  "states": [
    {
      "id": "<snake_case>",
      "description": "<one-sentence>",
      "rules": [
        {"trigger": "<NL statement>", "action": "<NL>"},
        ...
      ],
      "next": [{"to": "<state_id>", "on": "<NL event>"}]
    }
  ]
}
```