LEARNING TO LEARN EXPERIENTIAL KNOWLEDGE

Anonymous authors
Paper under double-blind review

ABSTRACT

Language agents can solve individual tasks effectively, yet they remain limited at accumulating *experiential knowledge*: converting prior task trajectories into reusable knowledge in free-form text that improves performance on future tasks. Prior work prompts strong models to write rules, skills, and strategies into an in-context notebook, but few approaches train models to improve this capability over long task streams. Training is challenging because the utility of each update is only revealed through later tasks. We study experiential knowledge accumulation through a persistent in-context notebook that is read by a *task model* and rewritten after each task by a *notebook model*. We propose NoteTaker, which trains the notebook model in two stages: SFT on teacher-generated notebook updates, followed by RL on short task streams, where each update receives credit only from subsequent task rewards that it could influence. Across three benchmarks, separately trained notebook models improve task models when deployed in online 40-task streams. These improvements transfer to held-out tasks, including AppWorld Test-Challenge tasks that involve applications and APIs absent from notebook model training. On AppWorld Test-Challenge, our finetuned Qwen3.5-9B notebook model increases task success from 0.0% to 25.8%, compared with 23.0% when Fable-5.1 serves as the notebook model. Analysis shows that SFT establishes basic experience abstraction and knowledge organization, while RL aligns notebook updates with downstream task performance, consistently improving over SFT across benchmarks and model sizes.

1 INTRODUCTION

Language model agents can solve individual tasks, but remain far behind humans in learning from experience and improving across tasks (Wang et al., 2025a). We study *experiential knowledge* accumulation: converting prior task trajectories into reusable, free-form textual knowledge. Prior work has shown the promise of persistent in-context notebooks, built by prompting strong models to abstract rules, skills, and strategies from trajectories (Zhao et al., 2024; Suzgun et al., 2026; Zhang et al., 2026d). However, few approaches explicitly train models to accumulate experience over long task streams. Such training is difficult because the value of notebook updates emerges only through subsequent task outcomes (Sutton, 1984; Pignatelli et al., 2024; Feng et al., 2025), and test-time task streams can be extremely long, making full-stream RL rollouts expensive (Fu et al., 2025; Lu et al., 2026).

We propose NoteTaker, a training framework for improving experiential knowledge accumulation. We study a setting where an agent maintains an in-context notebook that a *task model* reads before each task and a *notebook model* rewrites afterward, as shown in Figure 1. We freeze the task model and train only the notebook model. This isolates gains from better notebook updates from gains in task-solving ability, and it lets the trained notebook model pair with any task model. We optimize the notebook model in two stages. First, we apply SFT on teacher-generated notebook updates to teach basic behaviors for abstracting and organizing task experience. We then apply RL on short task-stream segments to align notebook updates with future task performance. Following turn-level credit assignment in multi-turn agent RL (Zhou et al., 2024; Feng et al., 2025), we credit each update with accumulated rewards from only the subsequent tasks it could have influenced, yielding a more informative learning signal.

We evaluate our training framework on three agentic benchmarks that span different forms of reusable cross-task knowledge: API usage patterns in AppWorld (Trivedi et al., 2024), navigation

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

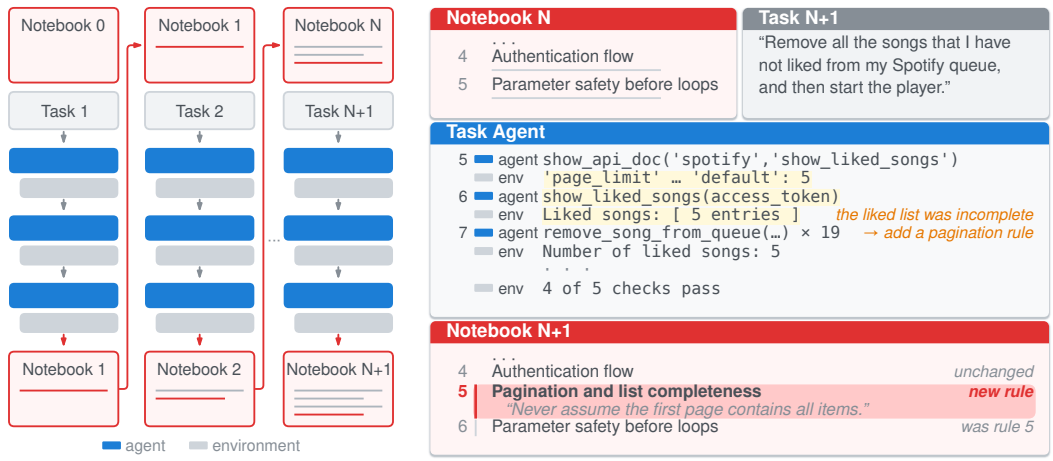


Figure 1: Experiential knowledge accumulation over a task stream. **Left:** The task model reads the persistent notebook before each task, and the notebook model rewrites it afterward using the trajectory and reward. **Right:** The notebook model preserves existing knowledge while turning a task-specific failure into a reusable rule.

and control strategies in MiniGrid (Chevalier-Boisvert et al., 2023), and embodied household procedures in ALFWorld (Shridhar et al., 2021). We train a separate notebook model for each benchmark and study generalization across task instances, stream lengths, and task models. We instantiate the task and notebook models with separate, size-matched Qwen3.5-4B or Qwen3.5-9B models (Qwen Team, 2026). Only the notebook model receives SFT and RL, while the task model remains frozen, allowing us to attribute changes in task success rate to experiential knowledge accumulation rather than updates to the task model. We also evaluate frontier and open-weight models as notebook models while keeping the corresponding task model frozen.

With a capable notebook model, a small frozen task model can perform substantially better: on AppWorld, Qwen3.5-9B reaches success rates of 36.5% on Test-Normal and 23.0% on Test-Challenge with Fable-5.1, versus 3.2% and 0.0% with an untrained, size-matched notebook model. This contrast shows that the task model can use experiential knowledge, while the untrained notebook model cannot reliably abstract it from task trajectories. Explicit training closes this gap: after SFT and RL, the Qwen3.5-9B notebook model reaches 34.0% and 25.8% on the two splits, respectively. Although RL uses eight-task continuations from a single task model, the trained notebook model remains effective over 40-task streams and transfers to trajectories from unseen task models. Our analysis shows that SFT establishes basic experience-abstraction and knowledge-organization behaviors, whereas RL aligns notebook updates with future task performance. In summary, our contributions are:

- We isolate a gap in *experiential knowledge*: models (Qwen3.5-4B and 9B) as agents can benefit substantially from high-quality notebooks, while they struggle to construct such notebooks from their own trajectories.
- We train dedicated notebook models using teacher supervision followed by downstream-reward RL, consistently improving over SFT across three benchmarks and two model sizes.
- We demonstrate robustness beyond the notebook model's training conditions: from eight-update training segments to 40-task deployment streams, from training tasks to held-out tasks, and from trajectories of the training task model to unseen task models.

More broadly, our workflow provides a foundation for training agents that operate over task streams to learn from prior experience, including settings with alternative knowledge representations or jointly trained task models.

2 Related Work

Continual Learning Agents. Continual learning traditionally studies how models improve over sequences of tasks through parameter updates while retaining previously acquired capabilities (Parisi et al., 2019; Kirkpatrick et al., 2017; Luo et al., 2025). Recent work extends this objective to language-model agents and evaluates whether experience from earlier tasks improves performance on later tasks (Wu et al., 2024; Zheng et al., 2025; Shu et al., 2026; Asawa et al., 2026). We study the same cross-task objective, but accumulate experience in token space rather than modifying model parameters (Zweiger et al., 2025) during a task stream. The model weights remain fixed within each task stream while task trajectories are converted into textual knowledge carried across tasks in a persistent notebook.

External Memory and Context Engineering. Prior work enables agents to carry information across interactions through two broad mechanisms. External-memory systems store structured records in databases and provide operations for adding, updating, deleting, and retrieving them (Park et al., 2023; Packer et al., 2023; Xu et al., 2025; Chhikara et al., 2025); CoALA frames such memory modules conceptually (Sumers et al., 2024), and long-term memory benchmarks evaluate them (Maharana et al., 2024; Wu et al., 2025). Context-engineering systems instead encode experience as textual artifacts, including verbal feedback, skills, workflows, playbooks, guidelines, manuals, and persistent documents (Shinn et al., 2023; Wang et al., 2024; Majumder et al., 2024; Zhao et al., 2024; Wang et al., 2025c; Fu et al., 2024; Chen et al., 2024; Fang et al., 2026; Suzgun et al., 2026; Zhang et al., 2026d). Even when revision is automated, the memory representation and update interface are generally specified through prompts, fixed operation vocabularies, or hand-designed retention rules (Kang et al., 2026; Chhikara et al., 2025; Zhang et al., 2026d; Zhou et al., 2026a), and even frameworks that evolve the memory architecture itself rely on LLM-driven redesign rather than training (Zhang et al., 2026b). Moreover, repeatedly rewriting a textual memory with an untrained LLM can make it less useful over time (Zhang et al., 2026a). Different from these methods, we train models over task streams to abstract experience into reusable knowledge and learn the update policy directly from downstream task performance.

Reinforcement Learning for External Memory. RL has been used to replace hand-written memory-update rules, but most prior work optimizes memory within a single task, either by updating external records for downstream question answering (Yan et al., 2026; Wang et al., 2025b; Yu et al., 2026b; Huo et al., 2026; Zhang et al., 2026c) or by rewriting context before the task produces its final outcome (Yu et al., 2026a; Zhou et al., 2026b; Zhang et al., 2026e; Li et al., 2026a; Yi et al., 2026; Lu et al., 2026; Li et al., 2026b). More closely related, several methods train memory update models over task streams but differ in how they assign task rewards to memory updates. Some broadcast a single outcome reward to every update (Ouyang et al., 2026), while others credit an update using only the next task (Cai et al., 2026) or a selected pair of tasks (Wang et al., 2026). The closest work is the concurrent CoD framework (Chen et al., 2026), which focuses on RL training for long-lifecycle agents. Our focus is complementary: we treat experiential knowledge accumulation as a distinct learning problem, freezing the task model and training a separate notebook model. This separation attributes performance gains to the notebook rather than to improved task solving, and we show that the learned update policy generalizes to trajectories from task models unseen during RL.

3 Method: NoteTaker

3.1 Task Formulation

We study how experiential knowledge accumulates over long streams of tasks. Our goal is a notebook model that converts task trajectories into reusable knowledge, so that task models perform better on future tasks. A *task stream* is a sequence of n tasks $(t^{(0)}, \ldots, t^{(n-1)})$ that share reusable information. For task k, a task model interacts with an environment and produces a trajectory $\tau^{(k)} = (o_1^{(k)}, a_1^{(k)}, \ldots, a_{H_k}^{(k)}, o_{H_k+1}^{(k)})$ and reward $r^{(k)}$. We treat post-task feedback as part of the terminal environment observation $o_{H_k+1}^{(k)}$ and hence as part of $\tau^{(k)}$. Although each task can be solved

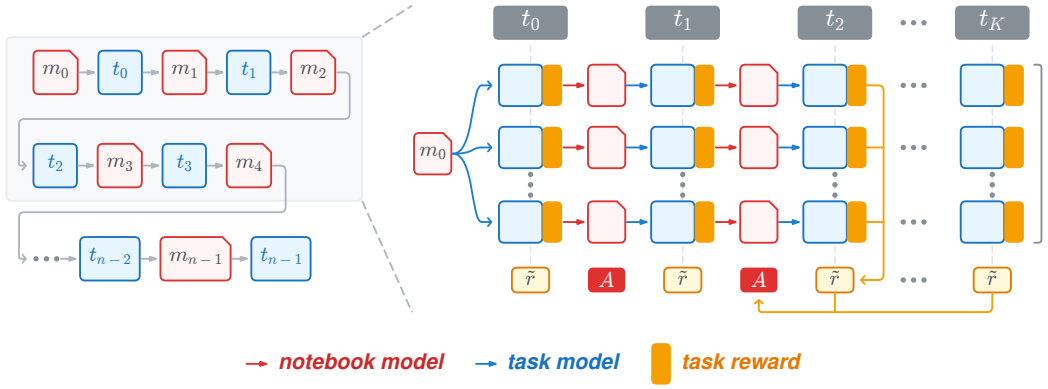


Figure 2: NoteTaker trains a notebook model on short segments of task streams for deployment on long task streams. Each notebook update receives credit only from rewards on tasks it can influence.

independently, our objective is to use experience from earlier tasks to improve performance on later ones.

A persistent text notebook carries information across tasks. Before task k, the frozen task model reads notebook $m^{(k)}$, starting from an empty $m^{(0)}$. After task k ends, the notebook model produces the complete revised notebook for the next task:

$$\hat{m}^{(k+1)} \sim \pi_\theta(\cdot \mid m^{(k)}, \tau^{(k)}), \qquad m^{(k+1)} = \mathcal{C}(\hat{m}^{(k+1)}). \tag{1}$$

The transition $\mathcal{C}$ enforces the fixed notebook budget. Because $m^{(k+1)}$ is written after $r^{(k)}$ is observed, it can affect only task $k+1$ and later tasks. The notebook is the only communication channel between tasks, and we train only π_θ while keeping the task model frozen. Figure 1 illustrates this causal boundary.

3.2 Teacher Supervision for Initialization

Small base models often struggle to turn raw task trajectories into reusable notebook content. We therefore initialize the notebook model with supervised examples from a stronger teacher, following the standard supervised cold-start stage before group-relative RL (Guo et al., 2025). We pair the teacher with the same frozen task model that the student notebook model will serve and run them over training task streams. After each task, the teacher maps the previous notebook and completed trajectory to a revised notebook. Each such rewrite provides a supervised example for the student notebook model, and the resulting checkpoint initializes RL. Section B gives the teacher, data, and optimization details.

3.3 Short-Stream RL with Causal Credit

Our goal is to maximize the frozen task model's average reward over a long task stream. Direct optimization requires expensive full-stream rollouts, so NoteTaker instead trains on short segments. Each training group begins with an empty notebook and an ordered sequence of $K+1$ tasks $(t^{(0)}, \ldots, t^{(K)})$. We sample G independent rollouts that share this initial notebook and task sequence. In rollout g, the task model first solves $t^{(0)}$ and produces $\tau_g^{(0)}$. At each position $k = 1, \ldots, K$, the notebook model rewrites the notebook from $(m_g^{(k-1)}, \tau_g^{(k-1)})$, after which the task model completes $t^{(k)}$ and receives reward $r_g^{(k)}$. Thus, each rollout contains K notebook updates and the K downstream rewards that can evaluate them. Sharing the task sequence aligns rewards by position, while independent sampling produces different notebooks and outcomes.

Position-Matched Causal Advantages A trajectory-level estimator averages the rewards in a rollout and assigns the same return to every notebook update. Instead, we compare rewards across rollouts at the same task position and propagate only causally subsequent rewards to each update.

Specifically, we compute a position-matched leave-one-out residual (Ahmadian et al., 2024) and its causal suffix return:

$$\tilde{r}_g^{(k)} = r_g^{(k)} - \frac{1}{G-1}\sum_{h\neq g} r_h^{(k)}, \qquad R_g^{(k)} = \sum_{\ell=k}^{K} \tilde{r}_g^{(\ell)}. \tag{2}$$

Position matching controls for differences in task difficulty, and the suffix ensures that an update receives credit only from rewards it could influence. This adapts the inner-group estimator of Feng et al. (2025) by using position within the task stream as the shared anchor, so the free-form notebook states need not repeat.

Let σ_R denote the pooled standard deviation of $R_g^{(k)}$ across all rollouts and task positions, computed with divisor $K(G-1)$. A single shared normalizer preserves the relative magnitude of returns across positions. Let $x_g^{(k)} = (m_g^{(k-1)}, \tau_g^{(k-1)})$ be the notebook-model input, $y_{g,t}^{(k)}$ its t-th output token, and $T_g^{(k)}$ its output length. We define the normalized advantage and token-level importance ratio together:

$$A_g^{(k)} = \frac{R_g^{(k)}}{\sigma_R + \epsilon}, \qquad \rho_{g,t}^{(k)}(\theta) = \frac{\pi_\theta(y_{g,t}^{(k)} \mid x_g^{(k)}, y_{g,<t}^{(k)})}{\pi_{\theta_{\text{old}}}(y_{g,t}^{(k)} \mid x_g^{(k)}, y_{g,<t}^{(k)})}. \tag{3}$$

We optimize the notebook model with the following clipped group-relative policy objective (Schulman et al., 2017; Shao et al., 2024):

$$\mathcal{J}(\theta) = \frac{1}{N}\sum_{g=1}^{G}\sum_{k=1}^{K}\sum_{t=1}^{T_g^{(k)}} \min\left(\rho_{g,t}^{(k)}(\theta)A_g^{(k)}, \text{clip}\left(\rho_{g,t}^{(k)}(\theta), 1-\epsilon_{\text{clip}}, 1+\epsilon_{\text{clip}}\right)A_g^{(k)}\right), \tag{4}$$

where $N = \sum_{g,k} T_g^{(k)}$ and ϵ_{clip} is the clipping threshold. The same update-level advantage $A_g^{(k)}$ is assigned to every token in a notebook update, so no learned value model is required.

4 Experimental Setup

4.1 Datasets and Evaluation Pipeline

We evaluate three types of agentic tasks: tool use in AppWorld (Trivedi et al., 2024), embodied household instruction following in ALFWorld (Shridhar et al., 2021), and grid-world navigation and control in MiniGrid (Chevalier-Boisvert et al., 2019; 2023). Although each task can be solved independently, tasks within a stream share environment rules, compositional skills, and planning strategies that can be reused across tasks.

For each benchmark, we construct fixed 40-task evaluation streams from its training and held-out splits. Because RL uses segments containing only eight notebook updates, the Train evaluation measures transfer to a longer horizon within the training distribution, while Test additionally measures transfer to held-out tasks. In AppWorld, Test (N) contains familiar applications and APIs, whereas Test (C) introduces unseen applications and more complex multi-application tasks. The task model interacts with the environment over multiple turns, after which the notebook model receives the completed interaction history. Every stream starts with an empty notebook and follows a fixed task order; we score each task by binary success and report the mean across the stream.

4.2 Baselines

We evaluate a range of models as notebook models on experiential knowledge accumulation. All notebook models share the same frozen task model (Qwen3.5-4B or 9B), notebook-model prompt, and 16KB notebook size limit. As a baseline, we evaluate each task standalone without a notebook, denoted *No Memory*. We then evaluate two proprietary notebook models, GPT-5.6-Sol and Fable-5.1, and one open-weight model, Kimi-K2.6, which also serves as the teacher in our training. Finally, we train notebook models initialized from Qwen3.5-4B and 9B, matching the task model size, and compare them against their untrained base models. We first fine-tune each base model on teacher-generated notebook updates (*SFT*) and then apply RL on top (*SFT+RL*).

Table 1: **RL substantially improves small-model notebook updaters across three benchmarks.** We report task success rate (%) on training and held-out task streams, with our SFT+RL updaters in bold. For AppWorld, N and C denote Test-Normal and Test-Challenge. The final columns average the per-benchmark scores, first averaging the two AppWorld test splits.

		AppWorld			ALFWorld		MiniGrid		Average	
	Updater	Train	Test (N)	Test (C)	Train	Test	Train	Test	Train	Test
Qwen3.5-4B	No memory	0.0	0.0	0.0	68.1	40.7	30.2	19.1	32.8	19.9
	GPT-5.6-Sol	24.2	18.0	10.8	90.6	78.1	54.5	34.5	56.5	42.3
	Fable-5.1	28.2	18.5	16.8	95.6	90.9	64.2	49.4	62.7	52.6
	Kimi-K2.6 (SFT teacher)	13.8	9.2	8.5	83.9	79.6	82.2	31.5	60.0	40.0
	Qwen3.5-4B	0.0	0.0	0.2	81.2	56.2	38.0	28.8	39.8	28.4
	+ SFT	15.2	5.2	4.5	87.5	77.1	42.9	30.1	48.6	37.4
	+ SFT + RL	**34.0**	**22.5**	**11.5**	**90.6**	**82.4**	**55.5**	**40.9**	**60.0**	**46.8**
Qwen3.5-9B	No memory	0.8	1.0	0.2	88.1	50.7	47.1	34.9	45.3	28.7
	GPT-5.6-Sol	34.2	24.2	19.5	91.9	85.3	61.2	47.0	62.5	51.4
	Fable-5.1	45.2	36.5	23.0	96.9	95.7	71.0	59.5	71.0	61.7
	Kimi-K2.6 (SFT teacher)	33.2	24.8	10.2	88.9	64.4	90.5	68.5	70.9	50.1
	Qwen3.5-9B	0.0	3.2	0.0	90.6	65.6	48.0	33.6	46.2	33.6
	+ SFT	13.8	19.5	2.2	94.4	79.4	51.0	36.1	53.1	42.1
	+ SFT + RL	**41.0**	**34.0**	**25.8**	**95.0**	**87.6**	**59.8**	**44.0**	**65.3**	**53.8**

4.3 Training Details

For each benchmark and model size, we train a separate notebook model initialized from either Qwen3.5-4B or Qwen3.5-9B. The task models remain frozen. We first supervise each notebook model using updates generated by Kimi-K2.6 from trajectories produced by the corresponding task model. We then apply NoteTaker's RL stage using segments of $K = 8$ notebook updates and $G = 16$ independently sampled rollouts per group, with every segment starting from an empty notebook; additional optimization and inference settings are provided in Section B.

5 Results

We first evaluate whether learned notebook models improve frozen task models and whether these improvements generalize beyond the conditions used for training. Unless stated otherwise, each comparison uses the same frozen task model, so differences arise from the notebook model rather than the underlying task model.

5.1 NoteTaker Enables Small Models to Accumulate Experiential Knowledge

Small models can use experiential knowledge but struggle to produce it. Table 1 reveals a substantial gap between the task model's ability to use experiential knowledge and the notebook model's ability to produce it. With the same Qwen3.5-9B task model, solving tasks without a notebook yields success rates of 1.0% on AppWorld Test-Normal and 0.2% on Test-Challenge. Using the base Qwen3.5-9B as the notebook model provides little benefit, reaching 3.2% and 0.0%, whereas replacing it with Fable-5.1 raises performance to 36.5% and 23.0%. The task model can therefore exploit high-quality experiential knowledge, even when its untrained counterpart cannot reliably extract that knowledge from previous trajectories.

NoteTaker substantially improves trained notebook models. Supervised initialization with Kimi-K2.6 raises the 9B notebook model from 3.2% to 19.5% on Test-Normal and from 0.0% to 2.2% on Test-Challenge, but leaves substantial room for improvement. Adding NoteTaker's RL stage improves every held-out result over SFT by 5.3–23.6 percentage points and produces the strongest trained Qwen notebook model in every evaluation setting. On AppWorld, the 9B model reaches 34.0% on Test-Normal and 25.8% on Test-Challenge, approaching Fable-5.1 on

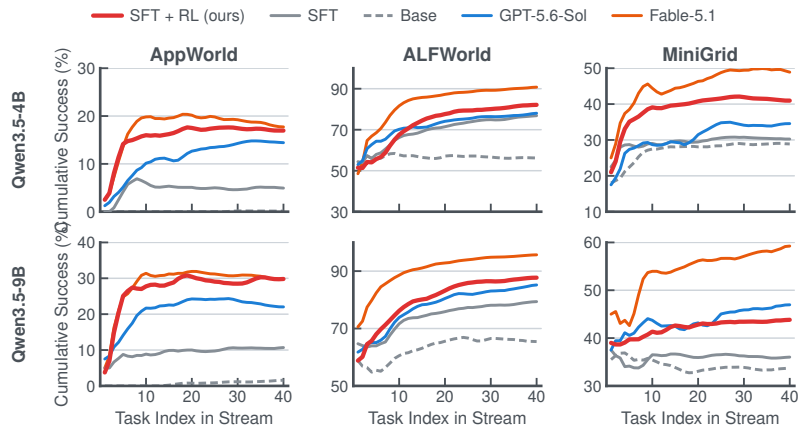


Figure 3: **NoteTaker-trained notebook models maintain their advantage throughout long deployment task streams.** Each curve reports task success averaged within the prefix from task 1 through task t, and then across held-out task streams.

Test-Normal and exceeding it on Test-Challenge. The same improvement over SFT appears for both model sizes on ALFWorld and MiniGrid, showing that the benefit is not specific to tool-use tasks.

Learned updating outperforms hand-designed context baselines. Our training objective is complementary to prior work on hand-designed memory representations and update prompts. Table 2 compares NoteTaker on AppWorld with a manually created static manual containing rules and a one-shot exemplar, and with ACE Online, an adaptive context-engineering method (Zhang et al., 2026d). NoteTaker outperforms both baselines for both model sizes on every task split, despite using an empty initial notebook and a minimal notebook update prompt. These results suggest that learning an update policy over task streams provides benefits beyond careful manual context design.

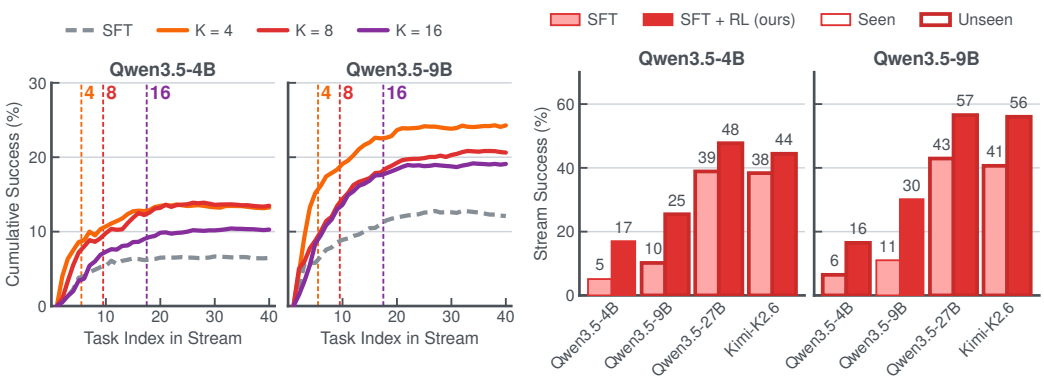
5.2 Notebook Models Generalize Beyond Training Conditions

The learned notebook policy remains effective beyond its RL training horizon. Figure 3 reports prefix-averaged success through each position of the 40-task held-out streams. Although RL trains the notebook model using only $K = 8$ updates, the SFT+RL model maintains an advantage over SFT and the base model throughout deployment on all three benchmarks and at both model sizes. The gap over SFT grows across the stream, which is consistent with continued use of newly observed trajectories rather than an advantage confined to the first few tasks. Figure 4(a) further evaluates notebook models trained with $K \in \{4, 8, 16\}$ at a matched optimization budget. For both model sizes and every value of K, prefix-averaged task success continues increasing beyond the corresponding training horizon, rising by 1.5–11.0 percentage points between that cutoff and task 40. The trained notebook update policy remains stable and effective when repeatedly applied far beyond its training horizon.

The notebook model transfers to unseen task models. During training, each notebook model observes trajectories from only one frozen, size-matched task model. Figure 4(b) shows that its improvement over SFT persists when it instead processes trajectories produced by the unseen Qwen3.5-27B task model: success increases from 38.9% to 47.7% for the 4B notebook model and from 42.9% to 56.6% for the 9B notebook model. This transfer indicates that NoteTaker learns to extract reusable experiential knowledge rather than adapting only to the trajectory style of its training task model.

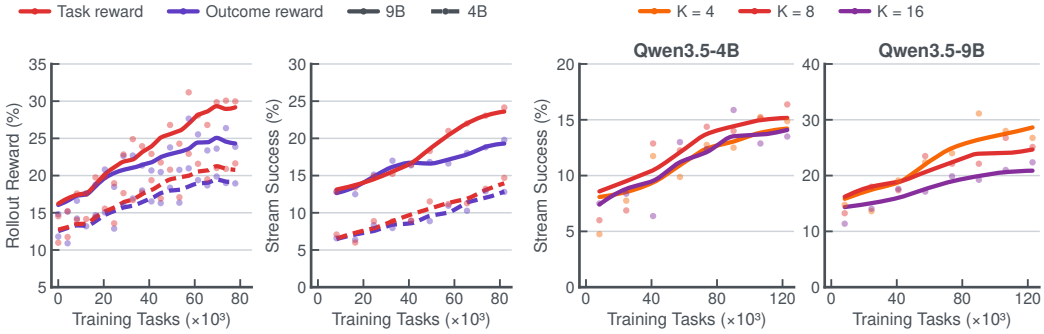
6 Analysis

We next analyze how credit assignment and rollout length affect RL and how different notebook models organize retained experiential knowledge.



(a) Stream length (b) Task model

Figure 4: **NoteTaker notebook models generalize beyond their RL training conditions on AppWorld. (a)** Prefix-averaged task success through task t for notebook models trained with $K \in \{4, 8, 16\}$. Dashed vertical lines mark the corresponding RL training horizons. **(b)** Task success rate when each notebook model processes trajectories from its training task model or unseen task models. Light and heavy bar outlines denote seen and unseen task models, respectively.



(a) Credit assignment (b) Rollout length K

Figure 5: **Credit assignment and rollout length affect RL effectiveness on AppWorld. (a)** Rollout reward and held-out task success rate during training with causal accumulated rewards or a trajectory-level outcome reward. **(b)** Held-out task success rate for rollout lengths $K \in \{4, 8, 16\}$ at matched numbers of sampled training tasks.

6.1 Effects of Credit Assignment and Rollout Length

Causal accumulated rewards improve credit assignment. Our method uses the accumulated rewards of causally subsequent tasks to evaluate each notebook update, as defined in Section 3.3. We compare this estimator with a trajectory-level alternative that assigns the same aggregate outcome to every update in the rollout. Figure 5(a) shows that the two estimators perform similarly early in training but diverge at later checkpoints for both model sizes. At the final matched checkpoint, our estimator raises the held-out AppWorld task success rate from 12.8% to 14.7% for the 4B notebook model and from 19.8% to 24.1% for the 9B notebook model. This result indicates that accumulated downstream rewards provide more informative credit when they are assigned according to which tasks each update could have influenced.

Rollout length presents a tradeoff. We next vary the number of consecutive notebook updates in each AppWorld training rollout, using $K \in \{4, 8, 16\}$. At matched numbers of sampled training tasks, Figure 5(b) shows that training with $K = 16$ progresses more slowly than with the shorter horizons. We do not expect a globally optimal K: a smaller value reduces sequential rollout cost but may miss longer-horizon effects and estimate notebook quality from too few downstream rewards, whereas a larger value provides a longer evaluation horizon but makes data collection more sequential and reduces the frequency of policy refreshes. We use $K = 8$ as an intermediate default and

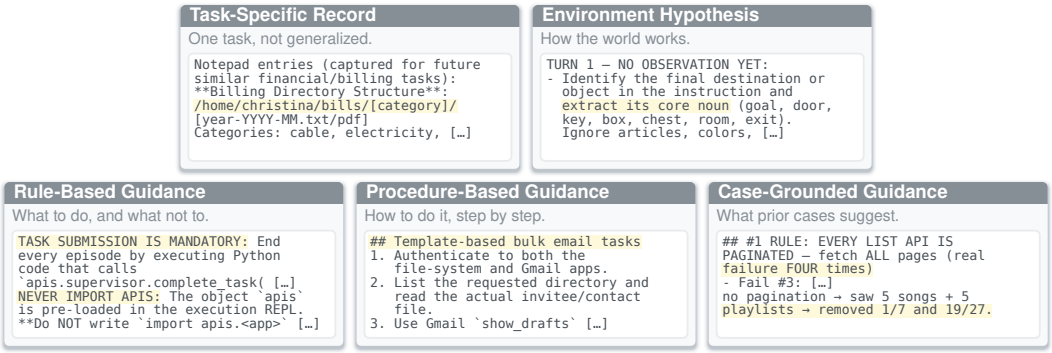


Figure 6: **Five categories for organizing retained experiential knowledge.** Each card shows a verbatim notebook excerpt illustrating one category. Categories can co-occur within a notebook and are not specific to any notebook model or benchmark.

treat this single-seed comparison as an illustration of the tradeoff rather than evidence for an optimal rollout length.

6.2 What Knowledge Do Notebook Models Retain?

A taxonomy of knowledge organization. We manually check the final notebooks from five notebook models (Base, SFT, SFT+RL, GPT-5.6-Sol, and Fable-5.1) on ten task streams from the test split from each of AppWorld, MiniGrid, and ALFWorld, holding the Qwen3.5-9B task model fixed. We define five categories that describe how notebooks organize retained experiential knowledge (see Figure 6). A *task-specific record* stores information about one task without generalizing it to other tasks. An *environment hypothesis* stores an observation or guess about the environment that may apply across tasks but does not tell the agent which actions to take. *Rule-based guidance* stores general instructions about which actions the agent should take or avoid. *Procedure-based guidance* stores an action sequence, high-level plan, or reusable template for completing a task. *Case-grounded guidance* stores previous task examples alongside guidance so the agent can reuse earlier behavior or judge how reliable the guidance is. Each notebook receives every applicable category tag rather than being forced into a single category.

Different notebook models favor different forms of organization. Across the 30 notebooks per model, GPT-5.6-Sol almost always writes rule-based (30/30) and procedure-based guidance (28/30) but never case-grounded guidance (0/30). Fable-5.1, in contrast, consistently writes case-grounded (30/30) and procedure-based guidance (30/30) but includes rule-based guidance less often (12/30). Thus, even strong models differ markedly in how they organize experiential knowledge. Compared with SFT alone on the same task orders, SFT+RL increases the presence of procedure-based guidance from 19/30 to 27/30 and rule-based guidance from 26/30 to 29/30, while case-grounded guidance remains absent (0/30). This suggests that RL barely introduces new forms of organization. Instead, it strengthens existing forms that better serve future tasks. Section D shows example notebooks from the five notebook models on the three benchmarks.

7 Conclusion

We establish experiential-knowledge accumulation as a distinct, trainable capability by freezing the task model and training only the notebook model. Small models can use strong notebooks but do not reliably create them without explicit training: NoteTaker combines SFT for reusable abstraction with RL for downstream utility, closing much of the gap to larger frontier models. Its update policies, trained on eight-task continuations, remain effective over 40-task streams and transfer to unseen task models. These results show that effective experiential learning depends not only on scale, but on learning what knowledge to retain.

AI USE STATEMENT

We used AI tools to help polish the manuscript by improving its clarity and organization. We also used them to discover and retrieve relevant literature, support research ideation, discuss experimental designs, and assist with coding, debugging, analysis, and visualization. The authors verified suggested references against the original sources and reviewed and tested all AI-assisted code. The authors are responsible for the final methodological choices, interpretation of the findings, accuracy of the references, and all content and artifacts in this paper.

REFERENCES

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to Basics: Revisiting REINFORCE-Style Optimization for Learning from Human Feedback in LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12248–12267, 2024. doi: 10.18653/v1/2024.acl-long.662.

Parth Asawa, Christopher M. Glaze, Gabriel Orlanski, Ramya Ramakrishnan, Benji Xu, Asim Biswal, Vincent Sunn Chen, Frederic Sala, Matei Zaharia, and Joseph E. Gonzalez. Continual Learning Bench: Evaluating Frontier AI Systems in Real-World Stateful Environments, 2026. URL https://arxiv.org/abs/2606.05661.

Yishuo Cai, Xingyu Guo, Xuancheng Huang, Jinhua Du, Can Huang, Wenxuan Huang, Wenhan Ma, Yuyang Hu, Aohan Zeng, Jie Tang, and Xu Sun. From Player to Master: Enhancing Test-Time Learning of LLM Agents via Reinforcement Learning over Memory. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. AutoManual: Constructing Instruction Manuals by LLM Agents via Interactive Environmental Learning. In *Advances in Neural Information Processing Systems*, volume 37, pp. 589–631, 2024. doi: 10.52202/079017-0019.

Yanxi Chen, Weijie Shi, Boyi Hu, Zeyue Zhang, Zhiwei Wang, Yuexiang Xie, Yaliang Li, Bolin Ding, and Jingren Zhou. Connect the dots: Training llms for long-lifecycle agents with cross-domain generalization via reinforcement learning, 2026. URL https://arxiv.org/abs/2606.20002.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: A Platform to Study the Sample Efficiency of Grounded Language Learning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJeXCo0cYX.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan K. Terry. Minigrid & Miniworld: Modular & Customizable Reinforcement Learning Environments for Goal-Oriented Tasks. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 36, pp. 73383–73394, 2023. doi: 10.52202/075280-3209.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory. In *ECAI 2025 – 28th European Conference on Artificial Intelligence*, volume 413 of *Frontiers in Artificial Intelligence and Applications*, pp. 2993–3000. IOS Press, 2025. doi: 10.3233/FAIA251160.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring Agent Procedural Memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17490–17502. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.866.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-Group Policy Optimization for LLM Agent Training. In *Advances in Neural Information Processing Systems*, volume 38, pp. 51797–51830, 2025. doi: 10.52202/085713-1544.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, Tongkai Yang, Binhang Yuan, and Yi Wu. AReaL: A Large-Scale Asynchronous Reinforcement Learning System for Language Reasoning. In *Advances in Neural Information Processing Systems*, volume 38, pp. 40748–40774, 2025. doi: 10.52202/085713-1218.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. AutoGuide: Automated Generation and Selection of Context-Aware Guidelines for Large Language Model Agents. In *Advances in Neural Information Processing Systems*, volume 37, pp. 119919–119948, 2024. doi: 10.52202/079017-3811.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, September 2025. doi: 10.1038/s41586-025-09422-z.

Yupeng Huo, Yaxi Lu, Zhong Zhang, Haotian Chen, and Yankai Lin. AtomMem : Learnable Dynamic Agentic Memory with Atomic Memory Operation, 2026. URL https://arxiv.org/abs/2601.08323.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A. Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. ACON: Optimizing Context Compression for Long-horizon LLM Agents. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. doi: 10.1073/pnas.1611835114.

Xiaozhe Li, Tianyi Lyu, Yizhao Yang, Liang Shan, Siyi Yang, Ligao Zhang, Zhuoyi Huang, Qingwen Liu, and Yang Li. Escaping the Context Bottleneck: Active Context Curation for LLM Agents via Reinforcement Learning, 2026a. URL https://arxiv.org/abs/2604.11462.

Yujiang Li, Zhenyu Hou, Yi Jing, Jie Tang, and Yuxiao Dong. CompactionRL: Reinforcement Learning with Context Compaction for Long-Horizon Agents, 2026b. URL https://arxiv.org/abs/2607.05378.

Miao Lu, Weiwei Sun, Weihua Du, Zhan Ling, Xuesong Yao, Kang Liu, and Jiecao Chen. Beyond the Context Window: Scaling Agentic RL via End-to-end Optimized Context Compression. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 21074–21125. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.966.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An Empirical Study of Catastrophic Forgetting in Large Language Models During Continual Fine-Tuning. *IEEE Transactions on Audio, Speech and Language Processing*, 33:3776–3786, 2025. doi: 10.1109/TASLPRO.2025.3606231.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating Very Long-Term Conversational Memory of LLM Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.747.

Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. CLIN: A Continually Learning Language Agent for Rapid Task Adaptation and Generalization. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=xS6zx1aBI9.

Siru Ouyang, Jun Yan, Yanfei Chen, Rujun Han, Zifeng Wang, Bhavana Dalvi Mishra, Rui Meng, Chun-Liang Li, Yizhu Jiao, Kaiwen Zha, Maohao Shen, Vishy Tirumalashetty, George Lee, Jiawei Han, Tomas Pfister, and Chen-Yu Lee. SkillOS: Learning Skill Curation for Self-Evolving Agents, 2026. URL https://arxiv.org/abs/2605.06614.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as Operating Systems, 2023. URL https://arxiv.org/abs/2310.08560.

German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual Lifelong Learning with Neural Networks: A Review. *Neural Networks*, 113:54–71, 2019. doi: 10.1016/j.neunet.2019.01.012.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative Agents: Interactive Simulacra of Human Behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, pp. 1–22. ACM, October 2023. doi: 10.1145/3586183.3606763.

Eduardo Pignatelli, Johan Ferret, Matthieu Geist, Thomas Mesnard, Hado van Hasselt, and Laura Toni. A Survey of Temporal Credit Assignment in Deep Reinforcement Learning. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=bNtr6SLgZf.

Qwen Team. Qwen3.5: Towards Native Multimodal Agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint arXiv:2402.03300*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. doi: 10.52202/075280-0377.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn.

Yiheng Shu, Bernal Jiménez Gutiérrez, Saisri Padmaja Jonnalagedda, Yuguang Yao, Huan Sun, and Yu Su. AgentCL: Toward Rigorous Evaluation of Continual Learning in Language Agents, 2026. URL https://arxiv.org/abs/2606.02461.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive Architectures for Language Agents. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=1i6ZCvflQJ.

Richard S. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst, 1984.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic Cheatsheet: Test-Time Learning with Adaptive Memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.eacl-long.333.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A Controllable World of Apps and People for Benchmarking Interactive Coding Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.850. URL http://dx.doi.org/10.18653/v1/2024.acl-long.850.

Ikechukwu Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, Joséphine Simon, Matthew Bennice, Chuyuan Fu, Cong Ma, Jiantao Jiao, Sergey Levine, and Karol Hausman. Jump-Start Reinforcement Learning. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 34556–34583. PMLR, 2023.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=ehfRiF0R3a.

Jiayin Wang, Zhiqiang Guo, Weizhi Ma, and Min Zhang. How Far Can LLMs Improve from Experience? Measuring Test-Time Learning Ability in LLMs with Human Comparison. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25677–25691. Association for Computational Linguistics, 2025a. doi: 10.18653/v1/2025.emnlp-main.1304.

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. Reinforcement Learning for Self-Improving Agent with Skill Library. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1529–1550. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.69.

Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. Mem-α: Learning Memory Construction via Reinforcement Learning, 2025b. URL https://arxiv.org/abs/2509.25911.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent Workflow Memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025c.

Cheng-Kuang Wu, Zhi Rui Tam, Chieh-Yen Lin, Yun-Nung Chen, and Hung-yi Lee. StreamBench: Towards Benchmarking Continuous Improvement of Language Agents. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 37, pp. 107039–107063, 2024. doi: 10.52202/079017-3398.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/d813d324dbf0598bbdc9c8e79740ed01-Abstract-Conference.html.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic Memory for LLM Agents. In *Advances in Neural Information Processing Systems*, volume 38, pp. 20004–20031, 2025. doi: 10.52202/085713-0593.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schütze, Volker Tresp, and Yunpu Ma. Memory-R1: Enhancing Large Language Model Agents to Manage and Utilize Memories via Reinforcement Learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12805–12825. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.583.

Lu Yi, Runlin Lei, Liuyi Yao, Yuexiang Xie, Yuyang Li, Wenhao Zhang, Zhewei Wei, Yaliang Li, and Jian-Yun Nie. Learning Agent-Compatible Context Management for Long-Horizon Tasks, 2026. URL https://arxiv.org/abs/2605.30785.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and Hao Zhou. MemAgent: Reshaping Long-Context LLM with Multi-Conv RL-based Memory Agent. In *International Conference on Learning Representations*, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/4264ee4376776907c0b87ed70b959585-Abstract-Conference.html.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic Memory: Learning Unified Long-Term and Short-Term Memory Management for Large Language Model Agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 21457–21483. Association for Computational Linguistics, 2026b. doi: 10.18653/v1/2026.acl-long.981.

Dylan Zhang, Yanshan Lin, Zhengkun Wu, Yihang Sun, Bingxuan Li, Dianqi Li, and Hao Peng. Useful Memories Become Faulty When Continuously Updated by LLMs, 2026a. URL https://arxiv.org/abs/2605.12978.

Guibin Zhang, Haotian Ren, Chong Zhan, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. MemEvolve: Meta-Evolution of Agent Memory Systems. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026b.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. MemSkill: Learning and Evolving Memory Skills for Self-Evolving Agents, 2026c. URL https://arxiv.org/abs/2602.02474.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models. In *International Conference on Learning Representations*, 2026d. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/8a94ff6f922d995d7d3f4ebf4143e442-Abstract-Conference.html.

Yuxiang Zhang, Jiangming Shu, Ye Ma, Xueyuan Lin, Shangxi Wu, and Jitao Sang. Memory as Action: Autonomous Context Curation for Long-Horizon Agentic Tasks. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 19149–19164. Association for Computational Linguistics, 2026e. doi: 10.18653/v1/2026.findings-acl.956.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM Agents Are Experiential Learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936.

Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, ZhongZhi Li, Yingying Zhang, Le Song, and Qianli Ma. LifelongAgentBench: Evaluating LLM Agents as Lifelong Learners, 2025. URL https://arxiv.org/abs/2505.11942.

Sizhe Zhou, Sheldon Yu, Hui Wei, Junda Wu, Siru Ouyang, Yizhu Jiao, Shijia Pan, Julian McAuley, Yu Zhang, Tong Yu, and Jiawei Han. Filesystem-Based Memory for LLM Agents: Organization, Evolution, and Sustainability, 2026a. URL https://arxiv.org/abs/2607.26637.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. ArCHer: Training Language Model Agents via Hierarchical Multi-Turn RL. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62178–62209. PMLR, 2024.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Bryan Kian Hsiang Low, and Paul Pu Liang. MEM1: Learning to Synergize Memory and Reasoning for Efficient Long-Horizon Agents. In *International Conference on Learning Representations*, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/5fc8b3bdfbb9167b5144df5d3fae4616-Abstract-Conference.html.

Adam Zweiger, Jyothish Pari, Han Guo, Ekin Akyürek, Yoon Kim, and Pulkit Agrawal. Self-adapting language models, 2025. URL https://arxiv.org/abs/2506.10943.

Table 2: **Comparison with static and adaptive context-engineering baselines on AppWorld.** We report task success rate (%). The static manual remains unchanged throughout each task stream, whereas ACE Online and NoteTaker update their context after every task.

Task model	Context method	Train	Test (N)	Test (C)
Qwen3.5-4B	Static manual	13.8	11.4	5.8
	ACE Online	21.5	14.9	5.9
	NoteTaker-4B	**34.0**	**22.5**	**11.5**
Qwen3.5-9B	Static manual	12.7	11.6	4.8
	ACE Online	34.1	25.5	12.5
	NoteTaker-9B	**41.0**	**34.0**	**25.8**

Table 3: **95% confidence intervals for the base and trained Qwen notebook models in Table 1.**

		AppWorld			ALFWorld		MiniGrid	
	Updater	Train	Test (N)	Test (C)	Train	Test	Train	Test
Qwen3.5-4B	Qwen3.5-4B	0.0±0.0	0.0±0.0	0.2±0.6	81.2±8.3	56.2±5.8	38.0±7.4	28.8±4.5
	+ SFT	15.2±10.1	5.2±4.2	4.5±2.4	87.5±5.5	77.1±4.0	42.9±6.9	30.1±4.8
	+ SFT + RL	**34.0**±4.1	**22.5**±3.3	**11.5**±6.5	**90.6**±5.1	**82.4**±3.5	**55.5**±8.2	**40.9**±5.9
Qwen3.5-9B	Qwen3.5-9B	0.0±0.0	3.2±3.7	0.0±0.0	90.6±5.4	65.6±6.4	48.0±7.8	33.6±4.9
	+ SFT	13.8±7.9	19.5±7.6	2.2±2.9	94.4±3.6	79.4±3.4	51.0±8.1	36.1±5.0
	+ SFT + RL	**41.0**±7.7	**34.0**±7.4	**25.8**±4.3	**95.0**±4.1	**87.6**±3.5	**59.8**±8.6	**44.0**±5.7

A Additional Results

A.1 Comparison with Static and Online Context Engineering

We evaluate two additional context baselines on AppWorld. The *static manual* is a manually created notebook containing general rules for AppWorld tasks and a one-shot exemplar. It initializes every task stream, is never updated, and contains no evaluation trajectories or labels. *ACE Online* starts from an empty context and applies ACE (Zhang et al., 2026d) after each task. These comparisons use the same frozen task models, notebook budget, task splits, and scoring protocol as NoteTaker.

A.2 Confidence Intervals for Main Results

For each cell, we compute the task success rate within each task stream and report the mean across streams. The $\pm$ values are half-widths of 95% confidence intervals computed from the variance of per-stream success rates using a Student-t distribution.

A.3 Training from Intermediate Notebook States

The RL stage of NoteTaker starts every rollout from an empty notebook and therefore never trains on the late-stage notebooks encountered in long test-time streams. To test whether this mismatch matters, we compare empty-start training (A) with rollouts initialized from notebooks produced by a Kimi-K2.6 teacher (B), an SFT notebook model (C), or an online replay pool (D), analogous to starting RL from states reached by a guide policy (Uchendu et al., 2023). All conditions use $K = 8$. Starting from intermediate states does not help: for the 9B notebook model (Table 4, left), empty-start training performs best, reaching 24.6% on Test (N) and 16.1% on Test (C), compared with 19.2–20.4% and 9.3–12.0% for the intermediate-state variants. Exposure to deployment-like states is therefore not sufficient on its own. Later states also yield weaker training signals. In condition C, whose starting depths are roughly balanced (Table 4, right), the fraction of no-gradient groups rises from 7.1–7.7% below depth 20 to 21.2–22.8% beyond it. All trajectories in these groups receive zero reward, indicating uniform failure rather than saturation. Notebook-cap events also rise, from 0.1% at depths 0–9 to roughly 12% beyond depth 20. Effective intermediate-state training may therefore require filtering starting states and trajectories, which we leave to future work.

Table 4: **Intermediate-state initialization does not outperform empty-start RL for the 9B notebook model.** The left table reports held-out task success rate and the overall percentage of no-gradient groups. The right table reports training coverage, no-gradient groups, and notebook-cap events by starting depth for condition C, initialized from the SFT library.

Condition	Test (N)	Test (C)	No gradient
SFT	19.5	2.2	–
A: empty	**24.6**	**16.1**	**0.0**
B: teacher	19.2	12.0	14.5
C: SFT library	20.4	9.3	14.8
D: online	19.6	11.7	27.6

Depth	Training coverage	No-gradient groups	Notebook-cap events
0–9	25.9	7.7	0.1
10–19	23.9	7.1	4.0
20–29	24.1	21.2	12.0
30–39	26.1	22.8	11.6

B Experimental Details

Supervised initialization. For each benchmark and model size, we train a separate notebook model initialized from either Qwen3.5-4B or Qwen3.5-9B. The notebook model and its corresponding task model are separate instances, and the task model remains frozen. To construct the supervised data, Kimi-K2.6 rewrites the notebook after each training task using the trajectory produced by the corresponding task model. We fine-tune each notebook model on its benchmark- and size-specific data for two epochs and use the resulting checkpoint to initialize RL.

Reinforcement learning. Each training group contains $G = 16$ independently sampled rollouts that start from an empty notebook and share an ordered sequence of $K + 1$ tasks. Every rollout contains $K = 8$ notebook updates and the eight subsequent task rewards used to score them. Each collection iteration contains 32 groups; with a global batch size of 512 notebook updates, this yields eight optimizer steps per iteration. All compared conditions use matched training and rollout seeds, with one independent training seed per condition.

Checkpoint selection and training budgets. Table 1 and Figure 3 use the same final checkpoints from the main training runs. Figure 4(a) compares all rollout-length conditions at optimizer step 144, providing a matched optimization budget across K. The intermediate-state comparison in Section A.3 uses checkpoints at optimizer step 96 for all conditions. All ablations use matched optimizer steps. We capped the ablation sweeps at the reported budgets because of their computational cost. They are intended as controlled comparisons rather than evidence of convergence. This explains why scores in Figure 4 and Figure 5 can differ from those in Table 1.

Optimization. We train in BF16 with Adam using a constant learning rate of 10^{-6}, $\beta_1 = 0.9$, $\beta_2 = 0.95$, zero weight decay, gradient-norm clipping at 1, and no dropout. We use a policy-ratio clipping threshold of 0.2, with no KL penalty, entropy bonus, or learned value model.

Rollout and inference settings. We sample notebook updates at temperature 1 with at most 32K prompt tokens and 32K generated tokens. The frozen task model runs at temperature 0 with hidden reasoning disabled, and its private reasoning is excluded from the notebook-model input. Each task-model response is limited to 4,096 tokens. Episodes allow at most 20 interaction turns in AppWorld and MiniGrid and 50 action turns in ALFWorld, terminating earlier upon success or environment termination. Section C specifies prompt construction, output processing, and the 16,384-byte notebook limit.

Evaluation task streams. Each evaluation stream contains 40 tasks. For AppWorld, we randomly sample tasks from the official Train, Test-Normal, and Test-Challenge splits, containing 90, 168, and 417 tasks, respectively. For MiniGrid, Train contains six semantic families comprising 47 environment variants, while Test contains five disjoint held-out families comprising 18 variants. Each stream uses one family and samples freshly seeded instances from its variants. For ALFWorld, Train contains simple pick-and-place, look-at-object-in-light, and clean-then-place tasks, whereas Test contains held-out heat-then-place, cool-then-place, and two-object placement tasks. Each stream uses one FloorPlan scene and samples eligible tasks without replacement. Task order is randomized rather than sorted by difficulty, so later positions are not intentionally easier than earlier ones.

C PROMPTS

Every prompt below is reproduced verbatim, including its role markers, markdown, and wording. Doubled-brace tokens such as `{{ notepad }}` are placeholders filled at runtime. The prompts use *notepad* for the persistent artifact called a notebook in the paper.

C.1 POST-TASK FEEDBACK

After the task model finishes, the environment first computes task success and then produces benchmark-specific feedback. We treat this feedback as the terminal environment observation in $\tau^{(k)}$. In the notebook-model prompt, it is serialized in the `Feedback on the attempt` field immediately after the visible interaction messages. The notebook is updated only afterward, so feedback from task k cannot affect its score and can influence only task $k+1$ and later tasks. The same feedback construction is used during training and evaluation.

C.2 NOTEBOOK MODEL

The same benchmark-agnostic prompt is used for the teacher, prompted notebook models, and trained notebook models.

Notebook model (rewrite mode)

```
You maintain a notepad for another agent that is working through a stream of tasks, one
after
another. The notepad is the only thing that carries over: the agent begins every future
task
with it and with nothing else it learned. You are judged solely on whether the notepad
you
produce raises that agent's success rate on **future, unseen tasks** --- not on how well
it
describes what just happened.

Think it through before you write. Work out what actually happened, whether the cause was
specific to this one task or something that will recur, and what the agent would have
needed to
know in advance.

## What the agent did

A JSON list of messages in order. Roles may include `user`, `assistant`, `tool` and
`reasoning`.

<messages>
{{ messages_json }}
</messages>

## Feedback on the attempt

Whatever the environment reported. This may include test results, scores, a reference
solution, or free-form notes, and it may be empty.

<feedback>
{{ feedback }}
</feedback>

## The current notepad

Line numbers are shown **for reference only** --- use them to talk about what to change.
**Do not
copy the line numbers into your output.**

<notepad>
{{ numbered_notepad }}
</notepad>

Keep the complete resulting notepad focused and within {{ budget_limit }} {{ budget_unit
}}. If it
is near the limit, prune, merge, or compress older material before adding anything.

## Output
```

```
Return the complete notepad as it should be for the next task, and **nothing else** --- no
preamble, no explanation, no code fence, no line numbers. You may add, reword, reorganise,
merge or delete anything. If nothing was learned, return the notepad unchanged.
```

Slot contents. `{{ messages_json }}` contains the visible task interaction; hidden task-model reasoning is excluded. `{{ feedback }}` contains the environment feedback, and `{{ numbered_notepad }}` contains the current notebook with reference-only line numbers. `{{ budget_limit }} {{ budget_unit }}` specifies the 16,384-byte notebook limit.

Output handling. The visible reply becomes the complete notebook for the next task, while hidden reasoning is discarded. Code fences and line-number prefixes are removed, unsafe role or tag markers are neutralized, and the result is truncated to the notebook limit. An empty reply clears the notebook.

C.3 Task Models

The task model is frozen, and each benchmark uses the same task-model prompt across all notebook conditions. The templates below show the complete prompts before their placeholders are filled.

C.3.1 AppWorld

The prompt provides the notebook, interaction protocol, API-discovery calls, user information, and task instruction.

AppWorld task model

````
USER:
Your notepad is below. It holds everything you have learned from previous tasks. Read it
carefully and follow it throughout this task.

<<<NOTEPAD>>>
{{ notepad }}
</notepad>

USER:
I am your supervisor, and you are an AI Assistant whose job is to complete my day-to-day
tasks fully autonomously.

To do this, you will need to interact with app(s) (e.g., spotify, venmo etc) using their
associated APIs on my behalf. For this you will undertake a *multi-step conversation*
using a python REPL environment. That is, you will write the python code, the environment
 will execute it and show you the result, based on which, you will write python code for
the next step and so on, until you've achieved the goal. This environment will let you
interact with app(s) using their associated APIs on my behalf.

Here are three key APIs that you need to know to get more information

# To get a list of apps that are available to you.

```python
print(apis.api_docs.show_app_descriptions())
```

# To get the list of APIs under any app listed above, e.g. spotify

```python
print(apis.api_docs.show_api_descriptions(app_name='spotify'))
```

# To get the specification of a particular api, e.g. spotify app's login api

```python
print(apis.api_docs.show_api_doc(app_name='spotify', api_name='login'))
```

Each code execution will produce an output that you can use in subsequent calls. Using
these APIs, you can now generate code, that I will execute, to solve the task.
````

```
USER:
Using these APIs, now generate code to solve the actual task:

My name is: {{ main_user.first_name }} {{ main_user.last_name }}. My personal email is {{
 main_user.email }} and phone number is {{ main_user.phone_number }}.
Task: {{ instruction }}
```

C.3.2 MINIGRID

The prompt provides the notebook, available actions, task goal, and initial observation.

MiniGrid task model

```
USER:
Your notepad is below. It holds everything you have learned from previous tasks. Read it
carefully and follow it throughout this task.

<<<NOTEPAD>>>
{{ notepad }}
</notepad>

USER:
You are an exploration master that wants to finish every goal you are given. Every round
I will give you an observation, and you have to respond an action and your thought based
on the observation to finish the given task. You are placed in a room and you need to
accomplish the given goal with actions.

You can use the following actions:

- turn right

- turn left

- move forward

- go to <obj> <id>

- pick up <obj> <id>

- go through <door> <id>: <door> must be an open door.

- toggle and go through <door> <id>: <door> can be a closed door or a locked door. If you
 want to open a locked door, you need to carry a key that is of the same color as the
locked door.

- toggle: there is a closed or locked door right in front of you and you can toggle it.
Your response should use the following format:
Thought:
<Your Thought>

Action:
<Your Action>

Game goal for the entire episode:
{{ game_goal }}

USER:
Current observation for turn 1:
{{ initial_observation }}
```

C.3.3 ALFWORLD

The prompt provides the action protocol, notebook, task instruction, initial observation, and turn limit.

ALFWorld task model

```
SYSTEM:
You control an agent in the text-only ALFWorld household environment. On every turn,
return
exactly one command copied from the current observation's `valid_actions` list and
nothing else.
One response executes one environment action. You have at most {{ max_turns }} action
turns.
Do not include analysis, an `assistant:` or `action:` prefix, JSON, quotes, or a code
fence.

Typical commands navigate to a receptacle, open or close it, take an object, put an
object in or
on a receptacle, inspect inventory, look, examine an object under a light, or use an
appliance to
clean, heat, or cool an object. Object numbers matter. Always copy the complete command
exactly
from `valid_actions`; never invent a command or combine two commands in one response.

The task succeeds only when the requested final world state has been reached. Continue
until the
environment reports success. Failed or redundant actions consume turns.

The protocol above is fixed. Use the notepad below only for learned strategy.

Notepad:
{{ notepad }}

USER:
Task: {{ instruction }}

Initial observation:
```json
{{ observation_json }}
```

Return the first command only.
```

D NOTEBOOK EXAMPLES

Each panel follows a preselected held-out task order and shows the first 450 characters of the retained notebook after tasks 10, 20, 30, and 40. The same ordered tasks are used across notebook models within a benchmark, although their trajectories can diverge. Bracketed ellipses mark omitted suffixes; the complete notebooks are included in the artifact.

AppWorld – Base

After task 10 – first 448 of 1,727 chars

```
- Playlist listing returns partial results:
check total count or pagination (
show_playlist_library may not return all
playlists)
- Search is essential for completeness: use
search_playlists() with empty query if
show_playlist_library misses items
- Supervisor credentials pattern:
show_account_passwords returns creds stored by
supervisor; may only show selected apps, not
all
- Spotify login token: credentials from
supervisor work but may need to
[...]
```

After task 20 – first 448 of 2,376 chars

```
**Status: Split expenses (taxi $60, food $128)
among roommates. Key lesson: divide by (1 +
roommates), create request for 3 people not
just 1.**

If supervisor credentials blocked, check
account list with show_account_passwords
directly.
For file_system, path '~/documents' resolves to
 '/home/[user]/documents'.
When 401 error occurs, re-authenticate
immediately; tokens often expire per-session.
Always verify API parameters in show_api_doc
before
[...]
```

After task 30 – first 449 of 2,207 chars

```
**VENMO PAYMENT TASK WORKFLOW (CRITICAL)**

Send Money Task:
1. Identify recipient IDs: `private_data.
relative_user_ids` OR use `apis.venmo.
search_friends()` then examine results for
friend IDs
2. Payment method priority: Check supervisor
for cards with funds → use existing card →
attempt refill with valid card
3. Send money: `apis.venmo.create_transaction(
to_user_id, amount=20, description=For Lunch)`
4. Auth token: Store from login response,
[...]
```

After task 40 – first 432 of 4,296 chars

```
<notepad>
SIMPLE NOTE HABIT EXPORT WORKFLOW (CRITICAL:
PARSE + FILE WRITE + ANSWER CAPTURE)

1. **Credentials**:
   - `apis.supervisor.show_account_passwords()`
 → Array with `account_name` and `password`
   - Extract password matching app name

2. **Auth Token (Critical)**:
   - SimpleNote `apis.simple_note.login()`
expects `username` (email), not `account_name`
   - Verify via `apis.api_docs.show_api_doc(
app_name='simple_note',
[...]
```

AppWorld – SFT

After task 10 – first 448 of 4,383 chars

```
- You have a hard step limit (commonly 20).
Budget your calls and make sure the very last
action is `apis.supervisor.complete_task(answer
=...)`. Never end with a `print` or
verification step.
- If the user asked a question expecting a
specific name, value, list, or entity, pass `
answer` with exactly that---no explanatory
sentences, no extra statistics, and no
additional prose unless explicitly requested.
- If the user issued an imperative command
[...]
```

After task 20 – first 450 of 8,172 chars

```
- You have a hard step limit (commonly 20).
Budget your calls and make sure the very last
action is `apis.supervisor.complete_task(answer
=...)`. Never end with a `print` or
verification step.
- If the user asked a question expecting a
specific name, value, list, or entity, pass `
answer` with exactly that---no explanatory
sentences, no extra statistics, and no
additional prose unless explicitly requested.
- If the user issued an imperative command ---
[...]
```

After task 30 – first 449 of 12,995 chars

```
- Keep reasoning and code blocks concise. The
assistant token limit is 4096; exceeding it
truncates the code block and aborts the task
with 0 mutations. Output that is too long
truncates (finish_reason='length'). Inside a
code block, never write step headers (e.g., '#
Step 1'), prose outlines, repeated commentary,
or speculative chains like Let's assume....
Only executable Python statements and ultra-
short comments are permitted; avoid verbose
[...]
```

After task 40 – first 443 of 8,123 chars

```
- **Hard token ceiling**: Capped at 4096 tokens
 per turn; truncation aborts with zero
mutations. Treat ~3000 tokens as the safe
maximum per code block. Every comment, string
literal, print statement, and step header
counts toward the limit. Split work across
turns (docs → data → compute → act) instead
of stuffing everything into one giant block.
- **Never repeat text inside a code block**:
Duplicated sentences, repeated fallback
commentary
[...]
```

AppWorld – GPT-5.6-Sol

After task 10 – first 448 of 13,118 chars

```
## API task workflow
- Invoke APIs directly as `apis.<app>.<api
>(...)`; do not import `apis` or app modules. `
api_docs` only documents APIs---it has no
generic `call_api`.
- Retrieve credentials with `apis.supervisor.
show_account_passwords()` and identity/email
with `apis.supervisor.show_profile()`.
Credentials are a list of records with `
account_name` and `password`; there is no `
app_name` field. The credential's `account_name
` may identify only
[...]
```

After task 20 – first 444 of 12,831 chars

```
## API task workflow
- Invoke APIs directly as `apis.<app>.<api
>(...)`; do not import app modules. `api_docs`
documents APIs but has no generic `call_api`.
- Inspect available apps and exact API docs
when names, parameters, pagination, response
shape, authentication, or side effects are
uncertain. If a guessed API name fails, use `
apis.api_docs.show_api_descriptions(app_name
=...)` rather than guessing repeatedly.
- Retrieve credentials with
[...]
```

After task 30 – first 437 of 16,292 chars; at 16KB cap

```
## API task workflow
- Invoke APIs directly as `apis.<app>.<api
>(...)`; do not import app modules. `api_docs`
has no generic `call_api`.
- Use `show_api_doc(app_name=..., api_name=...)
` for exact signatures, constraints, pagination
, and response shapes rather than guessing.
- Retrieve credentials with `apis.supervisor.
show_account_passwords()` and identity/email/
phone number with `apis.supervisor.show_profile
()`. Password records use
[...]
```

After task 40 – first 445 of 16,308 chars; at 16KB cap

```
## API task workflow
- Invoke APIs directly as `apis.<app>.<api
>(...)`; do not import app modules. `api_docs`
has no generic `call_api`.
- Use `show_api_doc(app_name=..., api_name=...)
` for exact signatures, constraints, pagination
, authentication parameters, and response
shapes rather than guessing.
- Retrieve credentials with `apis.supervisor.
show_account_passwords()` and identity/email/
phone with `apis.supervisor.show_profile()`.
Password
[...]
```

AppWorld – Fable-5.1

After task 10 – first 450 of 15,951 chars

```
# AppWorld agent notepad

## RULE 0 --- EVERY task ends with `apis.
supervisor.complete_task(...)` in a CODE cell
- A final prose message ENDS THE EPISODE with
no answer → graded as `<<not_given>>` → FAIL.
 This has now failed a task that was otherwise
5/5 correct (played the right song, then wrote
a summary instead of calling complete_task). It
 also sank an earlier Venmo question.
- Put `apis.supervisor.complete_task()` in the
SAME CELL as the final
[...]
```

After task 20 – first 447 of 16,166 chars; at 16KB cap

```
# AppWorld agent notepad

## RULE 0 --- EVERY task ends with `apis.
supervisor.complete_task(...)` in a CODE cell
- Ending with prose or running out of the 20-
cell budget = answer `<<not_given>>` → FAIL
even when every DB change is perfect (happened)
.
- `complete_task()` goes at the END of the same
 cell as the final action/verification ---
never a separate summary cell. COUNT YOUR CELLS
. From cell ~10 on, every acting/verifying cell
 MUST end with
[...]
```

After task 30 – first 445 of 15,333 chars

```
# AppWorld agent notepad

## RULE 0 --- EVERY task ends with `apis.
supervisor.complete_task(...)` in a CODE cell
- No `complete_task` (prose ending or 20-cell
budget exhausted) = answer `<<not_given>>` →
FAIL even when DB changes are perfect. HAPPENED
 TWICE: 20 cells burned on guessed params /
fallback `.get()` loops / forgotten
access_token.
- `complete_task()` goes at the END of the
final action+verification cell --- never a
separate summary
[...]
```

After task 40 – first 449 of 14,906 chars

```
# AppWorld agent notepad

## HELPER --- paste into the first working cell
 EVERY task (pagination bug caused 4 FAILS)
```
def all_pages(fn, **kw):
    out=[]; i=0
    while True:
        page=fn(page_index=i, page_limit=20, **
kw)
        out+=page
        if len(page)<20: return out
        i+=1
```
- EVERY list/search API (notes, groups,
contacts, transactions, tasks, recommendations,
 expenses...) defaults to 5 items, max 20. **A
result of exactly 5
[...]
```

MiniGrid – Base

After task 10 – first 445 of 890 chars

```
Action Format: Output exactly one valid action
string per turn. Invalid or duplicate responses
 waste a turn and reduce score.
Available Actions: Monitor the Available
actions list immediately. It determines valid
moves and reveals shortcuts (macros) like 'go
to goal'.
Obstacle Logic: If 'Door' or 'Wall' is in
observation, check specifically for 'toggle', '
open', or 'go through' actions. Primitive
movement may fail.
Turn Budget: 20-turn model
[...]
```

After task 20 – first 445 of 1,328 chars

```
Action Format: Output exactly one valid action
string per turn. Strictly match `
available_actions` list. If a macro (e.g. `go
to goal`) is not explicitly listed, do not
attempt it.

Observation Trust: Parse tool text for goal/
target cues (names, goal, exit). Trust
distances (1 steps away) over assumed state.
Correct hallucinations immediately based on
tool updates.

Navigation Orients: Do not move forward if
facing a wall. Turn left or
[...]
```

After task 30 – first 446 of 1,574 chars

```
Action Format: Output exactly one valid action
string per turn. Strictly match casing in `
available_actions` (e.g., move forward).
Action Efficiency: Do not query `
available_actions` unless the list is hidden or
 you are stuck. Assume primitives (`move`, `
turn`) are always available.
Movement Priority: `move forward` is the
default action. Only turn if facing a wall,
blocked path, or needing role alignment.
Wall Handling: When facing a wall,
[...]
```

After task 40 – first 444 of 2,000 chars

```
Goal Exit: Scan tool text for Goal, Win,
Completed, task is completed, or object name
change indicating success. Terminate
immediately if found.

Movement Priority:
(1) Only move forward if tool explicitly
indicates Goal is X steps away OR if facing
wall status is unrelated to goal (check context
).
(2) CRITICAL: If tool says barrier, cannot move
, or blocked, do NOT use move forward.
Immediately switch to turn left, turn
[...]
```

MiniGrid – SFT+RL

After task 10 – first 448 of 2,089 chars

```
GridGames environment: 20-turn episode limit.
Observations describe relative positions (e.g.,
 green goal 1 2 steps in front of you and 2
steps to your left). High-level macros such as
go to <object> and toggle and go through <door>
 are available and handle pathfinding
automatically.

Key lesson from a previous episode: The agent
got stuck in an infinite loop. It alternated
turn left and turn right when trying to face
the goal, but the
[...]
```

After task 20 – first 450 of 4,124 chars

```
GridGames environment: 20-turn episode limit.
Observations describe relative positions and
include an Available actions list. High-level
macros such as go to <object>, toggle and go
through <door>, and go through <door> are
available and handle pathfinding automatically.

Key lessons from previous episodes:
- alternating turn directions causes immediate
loops and wastes turns; always turn
consistently in one direction (e.g., turn left)
.
[...]
```

After task 30 – first 445 of 4,549 chars

```
GridGames environment: 20-turn episode limit.
Observations describe relative positions and
include an Available actions list. High-level
macros such as go to <object>, toggle and go
through <door>, and go through <door> are
available and handle pathfinding automatically.

CRITICAL: Every response must be exactly one
action string copied verbatim from the current
Available actions list. No extra text, quotes,
brackets, reasoning, or
[...]
```

After task 40 – first 445 of 4,822 chars

```
GridGames environment: 20-turn episode limit.
Observations describe relative positions and
include an Available actions list. High-level
macros such as go to <object>, toggle and go
through <door>, and go through <door> are
available and handle pathfinding automatically.

CRITICAL: Every response must be exactly one
action string copied verbatim from the current
Available actions list. No extra text, quotes,
brackets, reasoning, or
[...]
```

MiniGrid – GPT-5.6-Sol

After task 10 – first 447 of 2,039 chars

```
GridGames navigation:
- Emit exactly one action per turn, using the
exact wording of a known or listed action.
- Evaluation may end after 20 model turns, and
score decreases with primitive steps. Act
decisively; avoid idle turns, unnecessary scans
, and reversible turn patterns.
- For a `reach the goal` task, issue `go to
goal` immediately, even if the initial prompt
provides no observation or action list. This
navigation macro can complete the
[...]
```

After task 20 – first 446 of 2,650 chars

```
GridGames navigation:
- Emit exactly one action per turn, using the
exact wording of a known macro or currently
listed action. Invalid actions waste a turn.
- Evaluation may end after 20 model turns, and
score decreases with primitive steps. Act
decisively; avoid idle turns, orientation scans
, unnecessary checks, and reversible turn
patterns.
- For any task phrased as reaching the goal/
green goal square, the first action must be `go
 to goal`,
[...]
```

After task 30 – first 449 of 3,245 chars

```
GridGames navigation:
- Emit exactly one action per model turn, using
 the exact wording of a known macro or
currently listed action. Invalid actions waste
a turn.
- Episodes may end after 20 model turns, while
score also decreases with primitive steps.
Prefer a one-turn navigation macro over manual
movement, even if the macro expands into
several primitive steps.
- For `reach the goal`, `get to the green goal
square`, or equivalent goal-reaching
[...]
```

After task 40 – first 447 of 3,956 chars

```
GridGames navigation:
- Emit exactly one action per model turn, using
 the exact wording of a known macro or
currently listed action. Invalid actions waste
a turn.
- Episodes may end after 20 model turns, while
score also decreases with primitive steps.
Prefer valid one-turn navigation macros over
manual movement, even when a macro expands into
 several primitive steps.
- For `reach the goal`, `get to the green goal
square`, or equivalent direct
[...]
```

MiniGrid – Fable-5.1

After task 10 – first 442 of 10,445 chars

```
# GridGames navigation --- notepad

## TOP 3 RULES (violated repeatedly in early
episodes --- read first)
1. **NEVER alternate `turn left` / `turn right
`.** After a `turn left` that shows nothing,
the ONLY correct next scan action is another `
turn left`. `turn right` just returns you to a
view you already had (wasted 2--3 turns in ep.
2, 3, 6). Scan = same direction, max 3 turns.
2. **Use the macro in the `Available actions`
list the moment it
[...]
```

After task 20 – first 448 of 7,393 chars

```
# GridGames navigation --- notepad

## READ FIRST
20 episodes: 14 PASS / 6 FAIL. ALL 6 fails were
 single-room goal tasks with the goal out of
view. **2 of the 6 (ep. 17, ep. 20) were TOKEN-
LIMIT deaths**: after a navigation mistake the
model re-derived the whole map, ran past the
4096-token per-turn limit, emitted no action →
 instant FAIL. Every traverse task passed (6/6)
. Every blind `go to goal` opener that was
actually used won in 1 turn (7/9)
[...]
```

After task 30 – first 447 of 14,352 chars

```
# GridGames navigation --- notepad

## READ FIRST
30 episodes: 21 PASS / 9 FAIL. 8 fails = single
-room goal tasks with the goal out of view
after the scans. **The corner procedure below
has NEVER once been executed to the end in 8
tries --- every one of those 8 died on a
procedural slip, never on geometry:** ep. 17,
20 = token-limit deaths (re-derived the map
past the 4096-token per-turn limit → no action
); ep. 27 = FORMAT death (whole plan pasted
[...]
```

After task 40 – first 449 of 13,781 chars

```
# GridGames navigation --- notepad

## READ FIRST
40 episodes: 29 PASS / 11 FAIL. 10 fails =
single-room goal tasks with the goal out of
view after the scans; every one a procedural
slip, never geometry: ep. 17, 20 = token-limit
deaths (re-derived the map → no action); ep.
27 = FORMAT death (plan pasted into the reply,
re-sent 17×); ep. 29, 37 = walked toward the
biggest NUMBERED wall while a scan direction
had NO wall line; **ep. 40 = scan + walk
[...]
```

ALFWorld – Base

After task 10 – first 440 of 4,765 chars

```
KNOWLEDGE: COMMAND EXECUTION --- Use ONLY `
valid_actions`. Exact case, spacing, and syntax
 are mandatory. Do not infer variations.
KNOWLEDGE: TASK INTERPRETATION --- Follow user
intent strictly. If user suggests cool, select
cooling actions (`cool` with fridge). Do NOT
execute heating actions (`heat` with microwave
/) unless explicitly requested. Extraneous
transformations waste turns.
KNOWLEDGE: TOOL ACCESS SEQUENCE --- To use a
tool (e.g.,
[...]
```

After task 20 – first 450 of 1,815 chars

```
COMMAND EXECUTION: Use ONLY exact strings from
valid_actions. Case/spacing must match exactly.

LOCATION STRATEGY: Move directly to target
location before attempting object operations.
Never cycle between non-task locations without
purpose. Do not visit the same location twice
unless tool feedback indicates it may be
necessary.

OBJECT TRACKING: Complete the full object chain
 for each target. Track which objects have been
 moved to avoid redundant
[...]
```

After task 30 – first 448 of 1,599 chars

```
<notepad>
TASK SEQUENCE: Identify all verbs (e.g., heat +
 put). Execute all sequential requirements.
Process (transform) tasks must finish before
Place (move) tasks. Success requires all steps
completed.
SEARCH PRIORITY: If target surface is named in
the instruction (e.g., put on countertop),
CHECK THAT SURFACE FIRST. Do not default to
searching cabinets/fridges first. Items are
often already on the workspace or found there
faster.
SEARCH
[...]
```

After task 40 – first 447 of 1,808 chars

```
TASK LOGIC: Decompose goals. Action adjectives
imply backend verbs (e.g., cool -> `cool X with
 Y`). Perishables often originate in storage (
fridge/cabinet) requiring transport or
processing before final destination.
STATE TRACKING: Trust `valid_actions` over `
inventory` strings. `inventory` often reports
None while holding objects. Possession
confirmed when `valid_actions` explicitly lists
 `take` or `move` options.
EXECUTION: Commands must
[...]
```

ALFWorld – SFT

After task 10 – first 444 of 3,210 chars

```
- Only emit a single command that appears
verbatim in the current `valid_actions` list.
Repeating an invalid command or adding any
extra text consumes a turn without changing the
 environment. Confirm the command is present
this turn before emitting.
- `go to <location>` is only valid when
explicitly listed. Some nearby containers may
offer `open`, `examine`, or `take` without a `go
 to` step; if `go to` is missing but a useful
interaction is
[...]
```

After task 20 – first 448 of 4,083 chars

```
- Emit exactly one command, chosen verbatim
from the current turn's `valid_actions` list.
Never emit a command that is absent from the
list or add extra text. Invalid commands
consume a turn without any environmental effect
.
- `go to <location>` is valid only when
explicitly listed. However, `valid_actions` may
 include `open`, `examine`, `take ... from`, or
placement commands for adjacent containers or
nearby appliances without requiring you to
[...]
```

After task 30 – first 445 of 3,975 chars

```
- Emit exactly one command copied verbatim from
 the current turn's `valid_actions`. Never
invent, abbreviate, or retry a command that
returned invalid. **An invalid response means
that exact phrasing will not succeed this turn;
 choose a different exact match from the
current list and do not reuse the failed text
.**
- **Navigation rule:** `go to <place>` is only
valid when explicitly listed in `valid_actions
`. If `go to <place>` is absent but
[...]
```

After task 40 – first 450 of 5,190 chars

```
- Emit exactly one command copied verbatim from
 the current turn's `valid_actions`. Before
emitting, verify the string is present after
the latest environment transition. Never invent
, abbreviate, or retry a command that returned
invalid. An invalid response means that
phrasing is unavailable now and will not
succeed later unless the state changes; choose
a different exact match from the current list
and do not reuse the failed text.
- Navigation
[...]
```

ALFWorld – SFT+RL

After task 10 – first 445 of 3,215 chars

```
ALFWorld environment notes:

1. Action space syntax: Use only commands
verbatim from `valid_actions`. The command to
place an item is `move <item> to <receptacle>`.
 Never assume `put` or other synonyms work.

2. Microwave heating rule: To heat an item,
hold it and use `heat <item> with <microwave>`
while standing near the microwave. The
microwave may be closed for heating. Do NOT put
 the item inside the microwave first --- doing
so removes the
[...]
```

After task 20 – first 449 of 3,996 chars

```
ALFWorld environment notes:

1. Action space syntax: Use only commands
verbatim from `valid_actions`. The command to
place an item is `move <item> to <receptacle>`.
 Never assume `put` or other synonyms work.

2. Microwave heating rule: To heat an item,
hold it and use `heat <item> with <microwave>`
while standing near the microwave. The
microwave does not need to be opened; the `heat
` action is available even when the microwave
is closed. Do NOT
[...]
```

After task 30 – first 444 of 6,083 chars

```
ALFWorld environment notes:

1. Action space syntax: Use only commands
verbatim from `valid_actions`. The command to
place an item is `move <item> to <receptacle>`.
 Never assume `put`, `drop`, or other synonyms
work.

2. Inventory is unreliable and limited to one
object: The `inventory` field always shows None
 even when you are holding items. You can hold
only one object at a time. If `valid_actions`
contains item-specific commands for an
[...]
```

After task 40 – first 449 of 7,745 chars

```
ALFWorld environment notes:

1. Action space syntax: Use only commands
verbatim from `valid_actions`. Never assume `
put`, `drop`, or other synonyms work. If a
command you expect is not in `valid_actions`,
you are either at the wrong location or the
item is unavailable here. Do not guess commands
.

2. Inventory is unreliable and limited to one
object: The `inventory` field always shows None
 even when you are holding items. You can hold
only one
[...]
```

ALFWorld – GPT-5.6-Sol

After task 10 – first 449 of 4,895 chars

```
ALFWorld task-solving rules:
- Output exactly one complete command copied
verbatim from the current `valid_actions`.
Never issue a plausible command that is absent,
 omit a response, or add commentary; invalid
attempts consume decision turns without
changing state.
- Re-read `valid_actions` after every
transition. Being already at a location may
remove its `go to ...` action, while a nearby
fixture may be directly interactable (for
example, `open
[...]
```

After task 20 – first 448 of 5,916 chars

```
ALFWorld task-solving rules:
- Output exactly one complete command copied
verbatim from the current `valid_actions`.
Never issue a plausible command that is absent,
 omit a response, or add commentary; invalid
attempts consume decision turns without
changing state.
- Re-read `valid_actions` after every
transition. Being already at a location may
remove its `go to ...` action, while a nearby
fixture may be directly interactable.
- If a command is
[...]
```

After task 30 – first 447 of 6,793 chars

```
ALFWorld task-solving rules:
- Output exactly one complete command copied
verbatim from the current `valid_actions`.
Never issue a plausible command that is absent
or add commentary; invalid attempts consume a
decision turn without changing state.
- Re-read `valid_actions` after every
transition. Being already at a location removes
 its `go to ...` action, while nearby fixtures
may become directly interactable. If `open
cabinet 2` is listed but
[...]
```

After task 40 – first 447 of 8,739 chars

```
ALFWorld task-solving rules:
- Output exactly one complete command copied
verbatim from the current `valid_actions`.
Never issue a plausible command that is absent
or add commentary; invalid attempts consume a
decision turn without changing state.
- Re-read `valid_actions` after every
transition. Being already at a location removes
 its `go to ...` action, while nearby fixtures
may become directly interactable. If `open
cabinet 2` is listed but
[...]
```

ALFWorld – Fable-5.1

After task 10 – first 437 of 7,626 chars

```
# ALFWorld notepad (text household tasks, 50
decision-turn budget)

## Interface rules (violating these has cost me
 2-5 turns EVERY episode)
- Output exactly ONE command copied verbatim
from `valid_actions`. Anything else = Invalid
response, no state change, but it STILL burns a
 turn. Before sending, literally find the
string in the list.
- **If you get Invalid response, NEVER resend
the same string.** It will fail again. Re-read
[...]
```

After task 20 – first 445 of 10,323 chars

```
# ALFWorld notepad (text household tasks, 50
decision-turn budget)

## Interface rules (violating these has cost me
 2-5 turns EVERY episode --- read before EVERY
command)
- Output exactly ONE command copied verbatim
from `valid_actions`. Anything else = Invalid
response, no state change, but it STILL burns a
 turn. Before sending, literally find the
string in the list. If it isn't there, you are
probably in the wrong place --- the fix is
almost
[...]
```

After task 30 – first 449 of 15,663 chars

```
# ALFWorld notepad (text household tasks, 50
decision-turn budget)

## Interface rules (violating these has cost me
 2-5 turns EVERY bad episode --- read before
EVERY command)
- Output exactly ONE command copied verbatim
from `valid_actions`. Anything else = Invalid
response, no state change, but it STILL burns a
 turn. Before sending, literally find the
string in the list. If it isn't there, you are
probably in the wrong place --- the fix is
almost
[...]
```

After task 40 – first 447 of 14,996 chars

```
# ALFWorld notepad (text household tasks, 50
decision-turn budget)

## Interface rules (violating these has cost
2-5 turns EVERY bad episode --- read before
EVERY command)
- Output exactly ONE command copied verbatim
from `valid_actions`. Anything else = Invalid
response, no state change, but it STILL burns a
 turn. Before sending, literally find the
string in the list. If it isn't there, you are
in the wrong place --- fix is almost always `go
 to
[...]
```