LATENTGYM: A TESTBED FOR CROSS-TASK EXPERIENTIAL LEARNING WITH CONTROLLABLE LATENT STRUCTURE

Anonymous authors
Paper under double-blind review

ABSTRACT

We envision continually learning agentic systems that become more useful over time: as they encounter sequences of related tasks, they should infer the hidden structure shared across those tasks and use it to improve future decisions. This *cross-task experiential learning* capability is pivotal in domains such as personalization and interactive assistance, but existing training/evaluation frameworks do not provide shared, controllable latent structures and cannot measure whether or why agents improve. We introduce LATENTGYM: a controllable suite in which each environment is organized around a ground-truth latent variable governing the structure across tasks. Our construction yields agent-switching diagnostics that compare experience collection (how useful an agent's collected history is for subsequent performance) and experience utilization (how effectively an agent acts given an inherited history). We demonstrate our suite on empirical studies addressing three questions: how and why frontier models fail to adapt across related tasks; whether post-training on related task sequences improves cross-task adaptation, and where those gains come from; and how design choices such as inter-task feedback shape training dynamics and generalization. Together, these results establish a controlled foundation for studying how LLM agents learn from experience across tasks, and for designing agents that adapt more reliably in personalized, and interactive settings.

1 INTRODUCTION

We envision agentic systems that grow more capable the longer they work in a domain. An agent that encounters a sequence of related tasks, such as the tickets handled in customer support, the issues resolved across a software product, or the experiments run within a research area (Joshi et al., 2025; Lu et al., 2024), should improve with experience instead of solving each in isolation. It should pick up the shared structure across tasks, such as common knowledge, recurring patterns, and reusable strategies, and turn that experience into better decisions on the tasks that follow (Silver & Sutton, 2025). We call this ability *cross-task experiential learning*, the capacity to improve across a sequence of related tasks by recognizing what they share and acting on it. Humans do this routinely; whether today's LLM agents do, and how to build agents that do it reliably, remains largely open (Zheng et al., 2025a; Hu et al., 2026).

Rigorously studying whether agents improve across tasks, and why they fail when they do, is hard today. We can observe an agent's performance rise or fall over a sequence of tasks, but not easily say *why* a given failure occurred: whether the sequence carried too little shared structure to exploit, whether the agent failed to recognize the structure that was there, or whether it recognized the structure but failed to act on it. Nor can we cleanly compare methods meant to build this capability, since we have no setting in which the structure to be learned is known and controllable, making it hard to tell which method improves which part of the capability. We draw inspiration from how small, controlled testbeds have repeatedly driven the field forward by stripping away confounders, so that methods can be compared cheaply and their mechanisms understood. Toy environments such as MNIST and CIFAR in computer vision, and tabular gridworlds, control suites, and procedurally generated games in RL (Cobbe et al., 2020; Nikulin et al., 2023) allow researchers to build understanding before moving to messier and costlier real-world settings.

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.

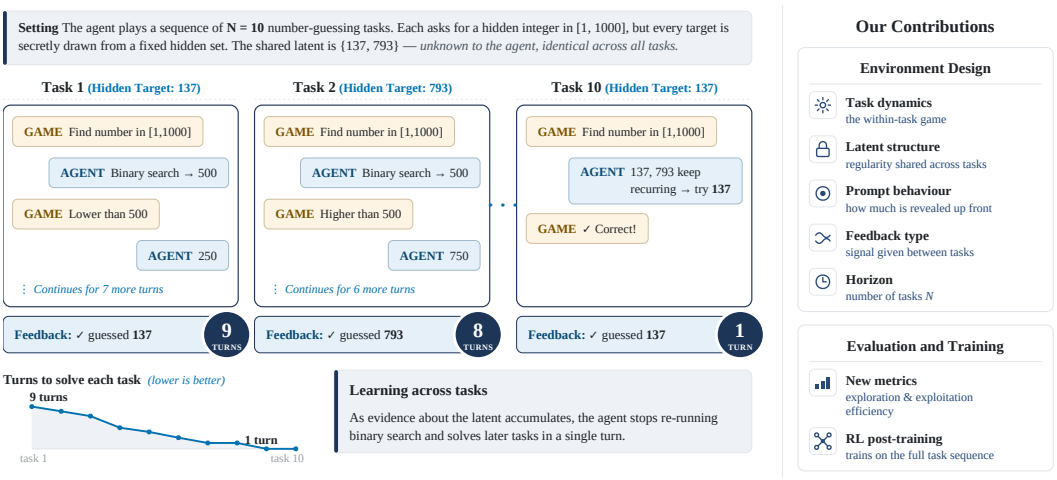


Figure 1. Probing Cross-Task Adaptation in LLM Agents through Controllable Latents. (Left) Number-guessing illustration. The agent plays a sequence of $N{=}10$ tasks; each task asks for a hidden integer in $[1, 1000]$, but targets are in fact drawn from a fixed hidden set $z = \{137, 793\}$ (the latent shared across tasks). Early on (Task 1), the agent runs binary search and takes 9 turns. By Task 10, having observed that 137 and 793 recur, it solves the task in 1 turn by guessing one of the recurring values directly. **(Right)** Each environment is built around such an explicit, controllable latent and factorizes along axes including task dynamics, prompt behavior, feedback type, and horizon.

We thus focus on small, fully controllable cross-task experiential learning problems where the experimenter knows the structure to be learned, runs are cheap enough to iterate on, and competing methods can be compared and understood. Such a testbed must make the structure shared across tasks both known to and controllable by the experimenter. We call this shared structure the *latent*: without ground-truth knowledge of it, an apparent failure to adapt cannot be interpreted. Figure 1 makes this concrete. The agent plays a sequence of number-guessing tasks. Each task asks for a hidden integer said to lie in $[1, 1000]$, but the targets are in fact drawn from a fixed hidden set $\{137, 793\}$, which is the latent shared across the sequence. Early on the agent has little evidence about the latent, so a near-optimal strategy is binary search over $[1, 1000]$. After a few tasks it can notice that the target keeps coming from a small set, and the later tasks test whether it acts on that evidence by guessing one of the recurring targets instead of restarting binary search. When the latent is known to the experimenter, they can see exactly where an agent's adaptation breaks down. We realize this testbed as LATENTGYM, a single framework combining controllable environments, diagnostics, and an integrated RL pipeline for cross-task experiential learning (Section 2).

Controllable environments. Each environment is organized around an explicit, controllable latent that defines the regularity shared across a sequence of tasks. Around this latent, each environment has a few additional independent knobs: the within-task dynamics, *prompt type* (how much information about the latent the prompt reveals), *feedback type* (what feedback the agent receives between tasks), and *horizon* (how many tasks the agent sees). Together, these knobs determine both how hard adaptation is and how it can be supported: the prompt sets the agent's prior over the latent and feedback governs how quickly evidence accumulates across tasks.

Diagnostics. Beyond measuring whether an agent improves over a sequence, LATENTGYM helps diagnose *how* it adapts. Two agents with the same cumulative reward may behave very differently: one keeps probing the latent while the other exploits what it already knows, and the two call for different remedies. Our agent-switching diagnostic separates *experience collection* (how useful an agent's collected history is for subsequent performance) from *experience utilization* (how effectively an agent acts given an inherited history).

An integrated RL pipeline. The same environments serve evaluation and training: they plug into a standard RL training stack through a thin adapter and expose per-task rewards across a sequence, so advantage functions can be defined over full sequences (Section 2).

Environments (including latents, prompts, feedback) and RL algorithms each registers independently and composes freely; adding a new one requires no changes to the rest. *Combining these into*

a single, extensible framework is the central contribution of this paper. We instantiate LATENTGYM with *seven environments* built on text-based games (Guertler et al., 2025). Using it, we report three findings, each made possible by the controllable latent and each carrying a concrete lesson for how agents learn across tasks:

1. **A capability evaluation of frontier models.** We evaluate frontier models (Claude Sonnet 4.6 (Anthropic, 2026), GPT-4o (OpenAI, 2024), Gemini 2.5 Flash (Google DeepMind, 2025)) and find that they can fail at cross-task adaptation even under simple latent structures (Section 3). Our framework surfaces three failure modes: *adaptation neglect* (failing to look for cross-task structure), *adaptation breakdown* (recognizing structure but failing to act on it reliably), and *adaptation miscalibration* (over-applying explicit information at the cost of task performance).
2. **Cross-Task RL improves cross-task adaptation.** We show that reliable adaptation can be induced by applying GRPO (Shao et al., 2024) over full sequences of related tasks, a recipe we call *Cross-Task RL* (Section 4). Performance gains mostly extend to out-of-distribution settings such as held-out latents and held-out environments. Further, agent switching shows that Cross-Task RL improves both experience collection and experience utilization, with the balance depending on the environment, switch point and reference agent.
3. **Algorithmic design choices in Cross-Task RL.** We study how feedback type and prompt type during training shape Cross-Task RL (Section 4). In our experiments, training with standard feedback transfers more robustly across evaluation conditions than training with full feedback.

Related Work Our framework directly exposes controllable latent structure and enables a systematic study of *cross-task* adaptation. On the other hand, in-context learning (Brown et al., 2020) and test-time scaling (Shao et al., 2024; DeepSeek-AI, 2025; Snell et al., 2025) largely consider a *single task*: demonstrations, reasoning traces, or retrieved context help the model do better on the current query, and each query is treated independently. Existing testbeds for LLM agents (Côté et al., 2018; Shridhar et al., 2021; Paglieri et al., 2025; Abdulhai et al., 2023; Liu et al., 2026; 2024; Jimenez et al., 2024; Zhou et al., 2024), including recent benchmarks for continual and experiential learning (Zheng et al., 2025a; Joshi et al., 2025; Zhao et al., 2024), measure capability or experience reuse on diverse tasks, and recent work explores harness design (Yao et al., 2023; Shinn et al., 2023; Wang et al., 2024) and meta-RL or continual-learning training for LLM agents (Jiang et al., 2026; Zheng et al., 2025b; Lin et al., 2026); in all of these, what should be learned across the sequence remains implicit and outside the experimenter's control.

Our setup inherits its formal structure from meta-RL (Beck et al., 2023), particularly the in-context variants where adaptation occurs through a sequence model's context (Duan et al., 2016; Wang et al., 2016; Mishra et al., 2018; Rakelly et al., 2019; Zintgraf et al., 2020) rather than through gradient updates (Finn et al., 2017). A closely related line studies whether transformers can internalize an RL algorithm in their forward pass (Laskin et al., 2023; Lee et al., 2023; Grigsby et al., 2024), by training on RL trajectories across many task families. Classical meta-RL benchmarks consider shared latent structure (Brockman et al., 2016; Open-Ended Learning Team et al., 2021; Bauer et al., 2023); the closest in spirit to ours is XLand-MiniGrid (Nikulin et al., 2023), which factorizes environments along clean axes. These benchmarks are non-linguistic and train small policies from scratch, saying little about pretrained LLM agents. We bring this paradigm into language space like Jiang et al. (2026); Lin et al. (2026) and develop a testbed that allows studying questions unique to LLM agents.

2 A CONTROLLABLE SUITE FOR CROSS-TASK EXPERIENTIAL LEARNING

An agent in LATENTGYM faces a sequence of N related tasks that share a hidden *latent*, the regularity common to all of them. A latent can be a rule, a preference, a mapping, a constraint, a target set, or a temporal pattern. The agent never observes it and must infer it from interaction. Its weights stay fixed across the sequence, so it adapts in-context, carrying the history of earlier tasks forward and conditioning on it, with no gradient updates. We score the per-task reward sequence $r_1, \ldots, r_N$ and summarize it by the *cumulative reward* $R = \sum_i r_i$; the final-task reward r_N and the improvement $r_N - r_1$ isolate other facets of adaptation. Because the latent governs every task, an information-seeking action that lowers an early r_i can raise many later ones, so the exploration–exploitation trade-off here runs *across* tasks, where evidence gathered early pays off on every task that follows.

Each environment in our framework is built around an explicit, controllable latent. Around this central axis, each environment factorizes along four further axes for controlled study: *core dynamics*

define the within-task decision problem; *task-specific variation* captures instance-level differences across tasks; *prompt type* controls how much information about the latent is revealed to the agent; and *feedback type* determines what information is added to the running history after each task.

Difficulty axis	**Set by**	**Example settings (number guessing)**
Within-task	Visible range	$[1, 100]$, $[1, 1000]$, $[1, 10000]$
Latent-identification	Size of hidden set	2, 5, 10
Cross-task	Latent + within-task difficulty	size 2 in $[1, 1000]$ vs. size 10 in $[1, 100]$

Table 1: Difficulty axes in the number-guessing environment.

Together, these axes let us independently control three orthogonal dimensions of difficulty (Table 1): *within-task difficulty* (how hard each task is in isolation), *latent-identification difficulty* (how hard the Latent is to infer from interaction), and *cross-task difficulty* (how strongly the latent governs subsequent tasks once inferred, jointly determined by the prior two). Varying the sequence horizon N controls how many tasks are available for adaptation. Prompt and feedback modulate these axes orthogonally: a richer prompt lowers latent-identification difficulty by revealing partial information about the latent, while richer feedback (e.g., revealing ground-truth outcomes regardless of success) accelerates how quickly evidence about the latent accumulates across tasks.

Composable and extensible software. The four axes are modular, swappable components, and not hard-coded options inside a monolithic environment. A fully specified environment is the product of five independently registered components,

$$\texttt{FullyDefinedEnv} = \text{core-env} \times \text{latent} \times \text{prompt} \times \text{feedback} \times N,$$

The registry composes these components into a runnable environment, identified by its configuration (e.g., number-guessing/set-of-3/no-info/standard/ep10). Each component can be varied independently, so each row of Table 1 is a different configuration rather than new code. A new environment is its single-task core dynamics and its registrations. The single-task dynamics conform to the standard `latentgym.core-env` interface, written from scratch or wrapped from an existing game. The latents, prompts, and feedbacks for that environment are declared in their own files and register themselves on import. From these registrations the full set of environments is composed automatically, and the shared machinery for composition, sequence generation, evaluation, and training is reused without change. Figure 5 in Appendix A shows the whole of number guessing in seven lines.

One environment object serves both evaluation and training: it implements the standard text-environment interface, reports per-task rewards and turn counts, and switching from evaluation to training changes only the operating mode. For training, a thin adapter exposes each environment to SkyRL (Cao et al., 2025), which runs the distributed rollouts and policy optimization and provides SFT, PPO (Schulman et al., 2017) and GRPO (Shao et al., 2024) out of the box; since per-task rewards are exposed across a sequence, new advantage estimators defined over full sequences plug in without changes to the rest of the stack. For evaluation, an inspection dashboard displays per-task reward curves and full trajectory conversations.

Diagnosing how an agent adapts. The reward curve $r_1, \ldots, r_N$ and its summaries $(R, r_N, r_N - r_1)$ tell us whether an agent improves over a sequence. They do not tell us *how*. Two agents with identical cumulative reward R can differ sharply in whether they probe the latent or exploit what they already know, and the two failures call for different fixes. Telling them apart is hard, because exploration and exploitation are entangled within a single run, and any late-task reward reflects both how informative the early interactions were and how well the agent acted on them. We investigate these contributions through an agent-switching diagnostic supported directly by the framework. One agent gathers the experience, and a different agent acts on it. At a switch point $K \in \{1, \ldots, N-1\}$, agent A plays tasks $1, \ldots, K$ and produces the accumulated history; control then passes to agent B, which inherits that same history and plays tasks $K+1, \ldots, N$. The framework preserves the full running context across the hand-off and attributes each task's reward to whichever agent played it, so a single run yields the post-switch (tail) reward $R_{\text{tail}}(A \to B) = \sum_{i=K+1}^{N} r_i$, where A explores for the first K tasks and B exploits over the remaining tasks. Agent switching compares experience collection and utilization by holding one role fixed at a reference agent $C \in \{A, B\}$. Swapping the initial agent while fixing the continuation agent measures how useful each agent's collected history is for subsequent performance. Swapping the continuation agent while fixing the initial

agent measures how effectively each uses the inherited history. We refer to these comparisons as exploration and exploitation gains, and report them relative to the reference agent's tail reward:

$$\text{ExploreGain}_C(B \text{ vs. } A) = \frac{R_{\text{tail}}(B \to C)}{R_{\text{tail}}(A \to C)} - 1, \qquad \text{ExploitGain}_C(B \text{ vs. } A) = \frac{R_{\text{tail}}(C \to B)}{R_{\text{tail}}(C \to A)} - 1.$$

The seven environments. We instantiate LATENTGYM with seven environments spanning different task dynamics and latent structures (Table 6), built on top of the TextArena suite of text-based games (Guertler et al., 2025). LATENTGYM differs in that each environment is a sequence of N games drawn from a shared latent structure: the agent plays multiple games of the same family (latent) in sequence and is rewarded for getting better across them. Each environment supports three prompt conditions (Table 7): no information about the latent, a vague hint that cross-task structure may exist, and full information about the latent. It also supports two feedback conditions: a *standard* condition, in which the agent receives only a binary success/failure signal after each task, and an *information* condition, in which the ground-truth is revealed regardless of whether the agent succeeded. Existing datasets such as SWE-Bench-CL (Joshi et al., 2025) evaluate learning across related tasks but do not explicitly control the shared structure or the benefit of knowing it (see Appendix A.11). More generally, isolating and varying this structure is difficult in complex tasks such as software maintenance. The seven environments in our suite provide a controlled setting for studying how agents discover shared structure and use it across tasks.

3 FAILURE MODES OF FRONTIER MODELS IN A MINIMAL SETTING

A controlled testbed is useful only if existing models struggle on it, exposing where they fall short. Our first demonstration establishes that LATENTGYM does this: the environments are deliberately simple, yet frontier models fail to learn across tasks on them, and they fail in interpretable ways, making it a discriminating place to develop and compare methods. We evaluate three representative frontier models (GPT-4o, Claude Sonnet 4.6, and Gemini 2.5 Flash) chosen to span different providers and capability profiles rather than to form an exhaustive leaderboard. These models fail at cross-task adaptation even under simple latent structures and across varying prompt conditions. Inspecting individual trajectories surfaces three recurring failure modes.

Adaptation neglect: models repeatedly restart from scratch. When models are not explicitly told that a hidden latent pattern exists, they often solve each task largely independently, without attempting to infer reusable structure from previous experience. This appears even in very simple latent structures, and we occasionally observe similar behavior even when models are told that such a pattern exists. In number guessing (latent: `set_of_3`), the target is always drawn from the same three values, yet all models repeatedly begin later tasks with the generic binary search strategy rather than testing the revealed targets. In the settings of Figure 7, none of the no-information runs shows reward improvement across tasks, suggesting that agents do not capably use the shared structure.

Claude Sonnet 4.6 solves each task independently: In each task, the agent identifies a hidden integer in $[1, 1000]$; after each guess, the game master indicates whether the target is greater, smaller, or equal. The shared **latent** restricts targets to $\{592, 781, 926\}$. Claude restarts binary search on every task in the ten-task sequence instead of using the recurring targets.

Task 1
Agent (turn 1): I'll use a binary search strategy. [500] … *continues for 6 more turns.*

Task 4
Agent (turn 1): Let me start with binary search again. [500] … *continues for 9 more turns.*

Task 9
Agent (turn 1): Let me start with binary search again. [500] … *continues for 9 more turns.*

Similar results appear across models and environments, see Figure 7 and Appendix D.1 for representative transcripts. Quantitatively, on `set_of_3` none of the three models ever opens a later task with one of the recurring targets under the no-information prompt (0% of tasks 6–10), and the number of guesses per task is the same on the first and the last task (Table 13). At first glance, this behavior may appear reasonable: a well-instructed agent should not impose hidden assumptions on a task. However, the pattern is more concerning in an adaptation setting. Real deployments contain recurring regularities that are not specified; an agent that only acts on explicitly stated structure will fail to improve with experience.

Adaptation breakdown: models notice structure but fail to use it reliably. When models are told that some pattern may exist, they often show partial but ineffective adaptation. Sometimes they ignore the hint and continue acting myopically (Appendix D.2.1); sometimes they search for a

pattern but fail to identify the correct one (Appendix D.2.3); sometimes they correctly infer part of the structure but do not act on it (Appendix D.2.2); and sometimes they try too hard to fit a pattern and drift away from the original objective of solving the task (Appendix D.2.4).

Gemini 2.5 Flash recognizes the pattern but does not act upon it: The agent guesses a hidden integer in $[365, 10365]$; the game master indicates whether the target is above, below, or equal to each guess. The **hidden latent** restricts targets to $[1400, 2400]$.

Task 7
Agent (turn 1): I've begun the process by defining the initial boundaries: 365 to 10365. My strategy remains binary search ... [5365] ... *continues for 12 more turns.*

Task 10
Agent (turn 1): I've reviewed the previous nine games. The numbers, ranging from 1461 to 2278, are well below the upper bound of 10365, although I'll continue using binary search ... [5365] ... *continues for 12 more turns.*

Table 13 quantifies this: with the vague cue, Claude Sonnet 4.6 and Gemini 2.5 Flash open later tasks with a recurring target in 100% and 96% of cases, yet Gemini still spends 6.8 guesses per task (Claude: 1.8), while GPT-4o does so in only 36% of tasks. The models are not simply incapable of forming hypotheses: in many trajectories, they explicitly propose plausible cross-task patterns. However, they do not know when a vague cue should be followed by further search, when enough evidence has been collected, or how strongly a hypothesis should influence future actions. More broadly, these cases suggest that recognizing latent structure and exploiting it are separable capabilities: a model can partially succeed at one while failing at the other.

Adaptation miscalibration: more information can make performance worse. Even when the relevant pattern is explicitly provided, models can suffer from miscalibrated adaptation and their performance might degrade relative to weaker description of the task (Figure 2). The explicit description appears to introduce a competing sub-goal: instead of using the latent as a calibrated prior, the model optimizes for demonstrating the stated pattern, over-applying the rule or choosing actions that satisfy its interpretation while violating the operational constraints of the game. Examples include proposing off-graph words in word ladder, choosing the wrong sub-range in number guessing, committing to a single arm in alternating bandits, or accepting too early in secretary-style tasks. Additional examples are shown in Figure 6; representative transcripts are in Appendix D.3. Across environments the effect is model-dependent: revealing the latent raises Gemini's average final reward ($0.62 \rightarrow 0.66$), leaves Claude's unchanged ($0.86 \rightarrow 0.85$) and lowers GPT-4o's ($0.78 \rightarrow 0.75$; more in Table 12). These cases show that in-context adaptation requires more than access to the right information: the model must also decide how strongly to condition on it.

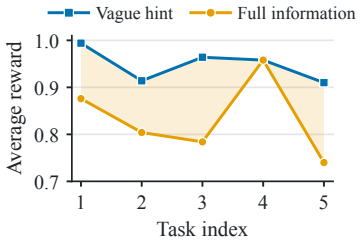


Figure 2. Full information can hurt adaptation. Gemini 2.5 Flash achieves lower reward on Word Ladder with the hub-word latent revealed than with a vague hint.

4 Cross-task RL for In-context Adaptation

Our second demonstration uses LatentGym to ask whether RL fine-tuning can instill general cross-task experiential learning. We compare two recipes. *Single-task RL* is the standard approach: fine-tune the policy on isolated tasks, with the reward depending on the reward r of that single task. *Cross-task RL* fine-tunes end-to-end on full sequence of N related tasks under a shared latent z, with the reward depending on the entire sequence of rewards $r_1, \ldots, r_N$. Under this signal, the policy is rewarded for using early tasks to infer the latent and act better on later ones. We address four research questions: **1.** Does cross-task RL induce adaptation, and how does it compare with single-task RL? **2.** Do the strategies learned by cross-task RL generalize out-of-distribution (OOD)? **3.** Does cross-task RL's lift come from collecting more useful experience, from using inherited experience better, or both? **4.** How do training-time feedback and prompt choices affect generalization?

Experimental Setup. We fine-tune Qwen3-8B and Gemma3-12B on fixed training latents, comparing the base instruction-tuned model with single-task RL ($N = 1$) and cross-task RL ($N = 10$, with accumulated context). We use Group Relative Policy Optimization (GRPO) (Shao et al., 2024) with a clipped objective and KL penalty against a frozen reference π_{ref}, optimizing cumulative reward $R = \sum_i r_i$. Evaluation reports cumulative reward, cross-task Gain, and final-task reward r_N, as defined in Section 2. We train both recipes to saturation until validation reward plateaus and additionally compare them under matched task-instance and episode budgets (Table 21). Table

uncertainties denote standard errors across five decoding seeds; a separate three-training-seed study assesses training stability (Table 23). Full training and evaluation details are in Appendix B. We report both model families in Table 2; for subsequent experiments, we present Qwen3-8B results in the main text and Gemma3-12B results in Appendix C.2. The reported training and evaluation runs used approximately 11,000 A100-80GB GPU-hours (Appendix B.6).

Cross-Task RL induces adaptation. We evaluate the base, single-task RL, and cross-task RL variants of both models on six environments. On cumulative reward $R = \sum_i r_i$ (Table 2), cross-task RL leads on every environment for both models, beating the base by an average of 99% (Qwen3-8B) and 83% (Gemma3-12B) and single-task RL by 43% and 55%; the largest gains over single-task RL are on Number Guessing for both models (+87% and +60%), together with Wordle for Qwen3-8B (+97%) and Secretary for Gemma3-12B (+130%). Single-task RL closes part of the gap only where the per-task problem is the main bottleneck (Word Ladder, within 8% of cross-task RL for both models; Secretary for Qwen3-8B and Hangman for Gemma3-12B), but trails cross-task RL elsewhere. The Gain column $(r_N - r_1)/r_1$ shows a similar picture of *cross-task* adaptation: cross-task RL is the variant with the most consistently positive Gain for both models (largest on Number Guessing, +83% and +43%, and Secretary, +23% and +39%), while single-task RL is negative on almost every environment (e.g., −57% and −55% on Wordle), indistinguishable from the base. Final-task reward r_N tracks the same ranking; cross-task RL improves r_N by an average of 59% (Qwen3-8B) and 74% (Gemma3-12B) over single-task RL, with Number Guessing gaining 2.4× for both models. These results show that cross-task RL induces stronger cross-task adaptation than single-task RL. Strong evidence that the policies use shared latent information comes from Number Guessing, where we can derive an upper bound on expected reward without previous-task information. On the shared two-number-set setting, Cross-Task RL achieves final-task rewards of 0.87 (Qwen3-8B) and 0.85 (Gemma3-12B), exceeding the history-blind bound of 0.8203 (see Appendix A.2). Conversely, when the same tasks are re-dealt so that no sequence shares a latent, with the prompt unchanged (Appendix C.6), the final-task reward of the same policies falls to 0.34 (Qwen3-8B) and 0.42 (Gemma3-12B) and the within-sequence gain disappears, so the excess over the bound is attributable to the shared latent rather than to generic in-context adaptation; the same pattern holds in the other environments (Table 24).

		Cross-task RL			Single-task RL			Base		
	Env.	Cum.	Gain (%)	Final	Cum.	Gain (%)	Final	Cum.	Gain (%)	Final
Qwen3-8B	Num. guess	**5.7** ±0.04	**+82.6** ±5.7	**0.64** ±0.01	3.1 ±0.03	−19.7 ±1.2	0.27 ±0.00	2.4 ±0.06	−20.2 ±2.0	0.20 ±0.01
	Mastermind	**7.2** ±0.02	**+6.6** ±0.9	**0.73** ±0.01	5.2 ±0.04	−19.7 ±0.9	0.50 ±0.01	4.4 ±0.04	−1.5 ±0.6	0.46 ±0.01
	Hangman	**6.4** ±0.04	**−1.7** ±1.7	**0.62** ±0.01	5.6 ±0.03	−3.4 ±0.7	0.57 ±0.01	3.5 ±0.02	+1.2 ±2.8	0.35 ±0.01
	Wordladder	**9.4** ±0.02	**+0.7** ±0.6	**0.94** ±0.00	8.7 ±0.03	−3.8 ±1.1	0.87 ±0.01	5.0 ±0.03	−6.9 ±2.9	0.50 ±0.01
	Secretary	**7.8** ±0.04	**+22.8** ±0.4	**0.80** ±0.00	7.1 ±0.01	+1.7 ±0.4	0.72 ±0.00	4.6 ±0.01	+15.8 ±0.6	0.46 ±0.00
	Wordle	**6.1** ±0.01	**+13.1** ±1.5	**0.62** ±0.01	3.1 ±0.01	−56.9 ±0.9	0.26 ±0.01	2.5 ±0.03	+8.3 ±3.2	0.26 ±0.01
Gemma3-12B	Num. guess	**6.0** ±0.04	**+43.2** ±1.5	**0.64** ±0.01	3.7 ±0.03	−48.0 ±2.6	0.26 ±0.01	4.0 ±0.03	+21.5 ±4.4	0.37 ±0.01
	Mastermind	**8.1** ±0.03	**−3.5** ±0.6	**0.76** ±0.01	5.5 ±0.06	−28.4 ±1.6	0.50 ±0.01	4.7 ±0.03	−11.4 ±2.0	0.47 ±0.01
	Hangman	**8.9** ±0.00	**+3.6** ±0.5	**0.89** ±0.00	8.6 ±0.01	−8.2 ±0.8	0.84 ±0.01	7.0 ±0.04	−12.1 ±0.6	0.66 ±0.01
	Wordladder	**9.6** ±0.01	**−1.0** ±0.5	**0.96** ±0.00	9.1 ±0.02	−5.3 ±0.7	0.90 ±0.00	6.9 ±0.02	−11.2 ±1.4	0.66 ±0.01
	Secretary	**9.7** ±0.00	**+39.0** ±0.0	**1.00** ±0.00	4.2 ±0.00	−1.8 ±0.1	0.42 ±0.00	4.4 ±0.01	+7.0 ±0.6	0.43 ±0.00
	Wordle	**6.7** ±0.01	**−13.9** ±0.5	**0.64** ±0.00	3.7 ±0.02	−55.4 ±0.7	0.33 ±0.01	2.3 ±0.03	−21.4 ±2.0	0.22 ±0.01

Table 2. Cross-task RL improves in-context adaptation. Comparison of Cross-task RL, single-task RL, and the base model across the six environments, for Qwen3-8B and Gemma3-12B. *Cum. Reward* is the total reward over the sequence ($R = \sum_i r_i$); *Gain* is the cross-task improvement from first to last as a percentage of the initial reward ($(r_N - r_1)/r_1$); *Final Reward* is the one on the last task (r_N).

To test whether a single policy can learn adaptation strategies across environments with different dynamics, we train on sequences from Number Guessing, Secretary, Hangman, and Word Ladder, with two latents each. Multi-environment cross-task RL outperforms multi-environment single-task RL on cumulative reward across all four environments, with average relative improvements of approximately 35% for Qwen3-8B and 28% for Gemma3-12B (Tables 3 and 14). Interestingly, multi-environment training can also sometimes improve on single-environment cross-task RL: for Qwen3-8B, cumulative reward increases by 20% on average, with the largest gain on Secretary (33%). Gemma3-12B shows no average improvement over its single-environment counterparts. These results suggest that one policy can support adaptation across environments, with joint training sometimes providing additional benefits.

Do strategies learned generalize beyond the training distribution? Next we test whether cross-task RL's gains generalize beyond the training distribution. A policy that has learned a transferable

		Cross-task (multi env)			Single-task (multi env)			Cross-task (one env)		
	Env.	Cum.	Gain (%)	Final	Cum.	Gain (%)	Final	Cum.	Gain (%)	Final
Qwen3-8B	Num. guess	**8.1** ±0.03	+21.2 ±0.6	**0.87** ±0.00	6.6 ±0.05	−5.5 ±1.6	0.62 ±0.01	6.3 ±0.05	**+54.2** ±4.5	0.64 ±0.01
	Hangman	**8.8** ±0.03	**+15.2** ±0.5	**0.90** ±0.00	5.5 ±0.04	+8.2 ±1.8	0.55 ±0.01	7.3 ±0.04	+12.9 ±0.7	0.75 ±0.01
	Wordladder	9.0 ±0.04	**+1.5** ±0.3	0.91 ±0.01	8.6 ±0.03	−5.4 ±1.1	0.86 ±0.01	**9.1** ±0.02	+1.1 ±1.0	**0.92** ±0.00
	Secretary	**9.5** ±0.00	+36.1 ±0.5	**0.96** ±0.00	6.3 ±0.01	−36.3 ±1.4	0.39 ±0.01	7.2 ±0.02	**+37.9** ±0.6	0.75 ±0.00

Table 3. Cross-task vs. single-task RL under multi-environment training. Cross-task RL improves cumulative reward over single-task RL, by approximately 35% on average (Qwen3-8B).

adaptation strategy should generalize beyond the specific latents and environments seen during training. We test this at two levels: held-out latents within trained environments (OOD-1), and entirely held-out environments (OOD-2).

For **OOD-1**, we hold out a latent within each environment and evaluate cross-task RL on sequences generated under those held-out latents: the environment is fixed, only the latent shifts. For Qwen3-8B, cross-task RL retains a clear advantage on cumulative reward in five of six environments (Table 4), beating the base by an average of 58% and single-task RL by 28%, with the largest gains over single-task RL on Wordle (+66%) and Number Guessing (+69%). The Gain column $(r_N - r_1)/r_1$ remains positive for cross-task RL in five of six environments, and cross-task RL achieves the highest Final Reward r_N in five of six environments. Gemma3-12B shows similar cumulative-reward benefits, outperforming the base in all six environments and single-task RL in five (Table 15).

		Cross-task RL			Single-task RL			Base		
	Env.	Cum.	Gain (%)	Final	Cum.	Gain (%)	Final	Cum.	Gain (%)	Final
Qwen3-8B	Num. guess	**6.2** ±0.08	**+47.3** ±14.9	**0.63** ±0.03	3.7 ±0.04	−6.7 ±3.3	0.35 ±0.01	2.8 ±0.04	−35.7 ±6.0	0.22 ±0.02
	Mastermind	**4.3** ±0.01	**+16.6** ±2.0	**0.44** ±0.01	3.5 ±0.03	−10.0 ±1.7	0.35 ±0.01	3.4 ±0.02	−6.0 ±3.3	0.37 ±0.01
	Hangman	**6.7** ±0.02	+26.8 ±1.1	**0.70** ±0.00	5.9 ±0.03	+16.5 ±1.3	0.60 ±0.01	5.6 ±0.03	**+69.1** ±8.3	0.59 ±0.00
	Wordladder	**9.6** ±0.02	−2.4 ±0.5	**0.94** ±0.01	9.3 ±0.02	**−0.0** ±1.4	0.92 ±0.01	7.1 ±0.03	−1.9 ±1.1	0.73 ±0.01
	Secretary	4.2 ±0.04	+5.3 ±1.3	0.39 ±0.00	**4.4** ±0.01	+6.0 ±1.3	0.42 ±0.00	4.4 ±0.02	**+40.3** ±2.8	**0.45** ±0.01
	Wordle	**4.9** ±0.03	**+1.8** ±3.1	**0.50** ±0.01	2.9 ±0.03	−49.5 ±0.5	0.26 ±0.00	2.0 ±0.03	−43.5 ±2.6	0.17 ±0.01

Table 4. Generalization under latent shift (OOD-1). Cross-task RL beats the base by an average of 58% and single-task RL by 28% on cumulative reward.

For **OOD-2**, we use a leave-one-out protocol over the same four environments as the multi-env experiment above: we train on sequences from three of them and test on the held-out fourth, so the policy has not seen the target environment in any latent during training. For Qwen3-8B, the leave-one-out model beats the base on cumulative reward on all four held-out environments by an average of 11%, with the largest gains on Word Ladder (18%) and Hangman (17%) (Table 5); r_N is also higher on every environment. The Gain results are mixed: the leave-one-out model beats the base on three of four environments but trails on Hangman, where the base shows a large positive Gain (+28%) driven partly by a low initial reward ($r_1 \approx 0.39$, vs. $r_1 \approx 0.56$ for the LOO model). For Gemma3-12B, cross-task RL improves cumulative reward on Number Guessing and performs similarly to the base on the other three held-out environments (Table 16).

		Cross-task RL (LOO)			Base		
	Env.	Cum.	Gain (%)	Final	Cum.	Gain (%)	Final
Qwen3-8B	Num. guess	**3.2** ±0.04	**−29.0** ±1.4	**0.25** ±0.01	3.0 ±0.06	−31.8 ±2.3	0.23 ±0.01
	Hangman	**5.6** ±0.06	+2.4 ±0.4	**0.57** ±0.01	4.8 ±0.05	**+28.3** ±2.8	0.50 ±0.01
	Wordladder	**4.9** ±0.06	**−1.5** ±2.0	**0.50** ±0.01	4.2 ±0.04	−11.4 ±3.7	0.42 ±0.01
	Secretary	**4.8** ±0.02	**+28.0** ±1.1	**0.49** ±0.00	4.8 ±0.01	+23.6 ±0.8	0.48 ±0.00

K	ExploreGain_C		ExploitGain_C	
	$C = \text{CT}$	$C = \text{ST}$	$C = \text{CT}$	$C = \text{ST}$
2	−0.6% ±0.3	+6.7% ±1.4	+28.0% ±0.4	+37.4% ±1.5
4	+0.1% ±0.4	+22.3% ±1.0	+14.5% ±0.5	+39.9% ±1.1
6	+5.0% ±0.5	+26.1% ±1.6	+12.5% ±0.5	+35.0% ±1.5
8	+9.8% ±0.9	+28.6% ±2.9	+12.3% ±1.0	+31.5% ±2.4

Table 5. (Left) Generalization under environment shift (OOD-2). A LOO cross-task model, trained on all environments *except* the evaluation one, beats the base on every OOD environment, with an average improvement of 11% on cumulative reward. **(Right) Exploration and exploitation gains on number guessing (Qwen3-8B).** Relative tail-reward gains (%) of Cross-task RL (CT) over Single-task RL (ST) at switch point K, with reference agent C held fixed, as defined in Section 2.

Since training optimizes cumulative reward $\sum_i r_i$, this is the primary metric on which we assess OOD generalization. Gains on this metric mostly persist under both OOD settings. We also observe improvements in Gain and final-task reward r_N, although less consistently across models and environments. Gain should be interpreted with caution, however: a model already strong on the initial task has limited room to improve (ceiling effect), while a model that is weak initially can post a large Gain simply from a low starting point (floor effect). Final Reward r_N should therefore be considered alongside Gain when interpreting generalization, and we do see cross-task RL's r_N exceed the base in almost all OOD cases for Qwen3-8B (five environments under OOD-1, all four under OOD-2).

Exploration vs. exploitation efficiency. To distinguish exploration from exploitation gains, we apply the agent-switching diagnostic of Section 2 to Number Guessing at $K \in \{2, 4, 6, 8\}$ (Figure 3). Writing CT and ST for Cross-Task and Single-task RL, we report $\mathrm{ExploreGain}_C(\mathrm{CT\ vs.\ ST})$ and $\mathrm{ExploitGain}_C(\mathrm{CT\ vs.\ ST})$ with the reference agent $C \in \{\mathrm{CT}, \mathrm{ST}\}$ held fixed (Table 5).

On Number Guessing, exploration gains range from -0.6% to $+28.6\%$ and increase with K for both reference agents (Table 5). The point estimates are positive in seven of eight settings; with $C = \mathrm{CT}$, the gain is slightly negative at $K = 2$ ($-0.6\% \pm 0.3$) and near zero at $K = 4$ ($+0.1\% \pm 0.4$). Exploitation gains are positive throughout, ranging from 12.3% to 39.9%. They decrease with K for $C = \mathrm{CT}$, but for $C = \mathrm{ST}$ peak at $K = 4$ before declining. The reported exploitation gain exceeds the exploration gain at every matched (K, C), although the gap narrows at later switch points. These results indicate a more consistent gain in experience utilization, with collection gains that depend on the switch point and reference agent. Both gains are relative, not pure: a collected history differs in content and presentation, and post-switch reward reflects task-solving ability and further exploration as well as use of the inherited history (Additional results in Appendix C.7).

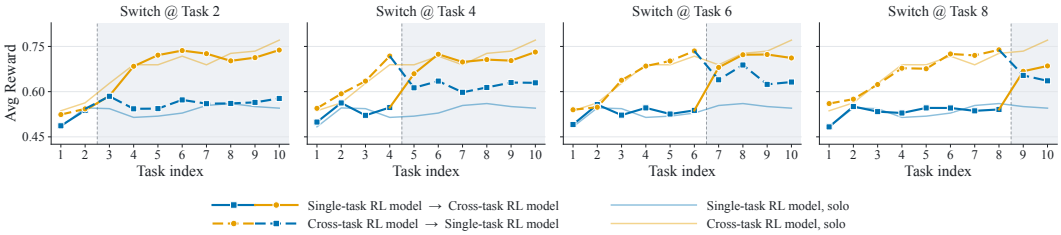


Figure 3. Agent-switching: cross-task RL vs. single-task RL on Number Guessing (Qwen3-8B). At a switch point $K \in \{2, 4, 6, 8\}$, control passes to another agent that inherits the cross-task context accumulated by the first. Solid/Dashed lines: trajectories of the two switch directions. Thin pale lines: each agent running the full sequence alone (*solo*).

Training-time feedback shapes generalization. We next ask how the *in-context feedback* used during training affects training dynamics and generalization. We train models under combinations of standard vs. full feedback between tasks and some-info vs. full-info prompt, while keeping the cumulative-reward objective $R = \sum_i r_i$ fixed, and evaluate each across all four prompt/feedback combinations at deployment. Averaged across evaluation conditions, standard-feedback training achieves higher reward than full-feedback training under both training prompt regimes (Figure 4). Standard-feedback training also shows greater robustness when evaluation feedback changes. Full-feedback-trained models drop by 0.509 points when the eval feedback switches to standard, whereas standard-feedback-trained models hold up (Appendix C.8). Prompt richness produces no substantive difference in aggregate, but the aggregate hides an asymmetry. Full-info-trained models score 0.36 higher on full-info eval than on some-info eval, whereas some-info-trained models are essentially indifferent to the eval prompt. Some-info training thus yields a more uniformly transferable model, at the cost of full-info training's advantage on full-info eval.

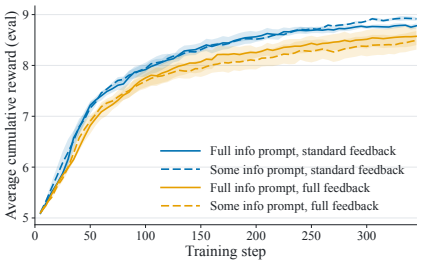


Figure 4. Evaluation reward during training. Standard feedback outperforms full one across training prompts; prompt type has a smaller effect.

5 Conclusion

We introduced LatentGym, a set of controllable language environments with explicit shared latents, for studying how past experience shapes agent behavior. Frontier models show interpretable adaptation failures, and sequence-level Cross-Task RL improves cumulative performance and induces shared-latent use in selected settings, although transfer varies by task and model. Agent-switching interventions offer a reference-dependent diagnostic of how experience is collected and used. Combining evaluation, diagnosis, and training, LatentGym supports controlled study of how agents discover and use shared structure across tasks.

AI USE STATEMENT

In this work, we used generative AI tools to create and edit software code (evaluation, aggregation and plotting scripts), to create and modify tables and figures. We have not used generative AI tools to propose or refine hypotheses, design the methodology or experiments, or interpret results; generating synthetic datasets, theoretical models, mathematical claims and proofs, translation, and qualitative analysis are not applicable to this work. We have reviewed all AI-assisted work: AI-generated code was verified and tested against independent implementations by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work involves no human subjects or personal data; all environments are synthetic text games. We release the environment suite, the training and evaluation pipeline, and RL fine-tunes of two open-weight models under their base licenses, and query frontier models through public APIs under the providers' terms. We see no direct path to harmful applications.

REPRODUCIBILITY STATEMENT

Environments and prompts are specified in Appendix A; training and evaluation protocols, hyperparameters (Table 8) and compute (Section B.6) in Appendix B. All results are mean $\pm$ standard error over five decoding seeds with 50 trajectories per cell; a training-seed study is in Appendix C. Code is available here.

REFERENCES

Marwa Abdulhai, Isadora White, Charlie Snell, Charles Sun, Joey Hong, Yi Zhai, Kelvin Xu, and Sergey Levine. LMRL gym: Benchmarks for multi-turn reinforcement learning with language models. *arXiv preprint arXiv:2311.18232*, 2023.

Anthropic. Claude Sonnet 4.6 system card, 2026. URL https://www.anthropic.com/claude-sonnet-4-6-system-card.

Jakob Bauer, Kate Baumli, Feryal Behbahani, Avishkar Bhoopchand, Nathalie Bradley-Schmieg, Michael Chang, Natalie Clay, Adrian Collister, Vibhavari Dasagi, Lucy Gonzalez, et al. Human-timescale adaptation in an open-ended task space. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023.

Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. A survey of meta-reinforcement learning. *arXiv preprint arXiv:2301.08028*, 2023.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Shiyi Cao, Dacheng Li, Fangzhou Zhao, Shuo Yuan, Sumanth R. Hegde, Connor Chen, Charlie Ruan, Tyler Griggs, Shu Liu, Eric Tang, Richard Liaw, Philipp Moritz, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. SkyRL-Agent: Efficient RL training for multi-turn LLM agent, 2025. URL https://arxiv.org/abs/2511.16108.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Ruo Yu Tao, Layla El Asri, et al. TextWorld: A learning environment for text-based games. In *Workshop on Computer Games at IJCAI*, 2018.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.

Google DeepMind. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities, 2025. URL https://storage.googleapis.com/deepmind-media/gemini/gemini_v2_5_report.pdf.

Jake Grigsby, Linxi Fan, and Yuke Zhu. AMAGO: Scalable in-context reinforcement learning for adaptive agents. In *International Conference on Learning Representations (ICLR)*, 2024.

Leon Guertler, Bobby Cheng, Simon Yu, Bo Liu, Leshem Choshen, and Cheston Tan. TextArena. *arXiv preprint arXiv:2504.11442*, 2025.

Qisheng Hu, Quanyu Long, and Wenya Wang. When continual learning moves to memory: A study of experience reuse in LLM agents. *arXiv preprint arXiv:2604.27003*, 2026.

Yulun Jiang, Liangze Jiang, Damien Teney, Michael Moor, and Maria Brbic. Meta-RL induces exploration in language agents, 2026. URL https://arxiv.org/abs/2512.16848.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Thomas Joshi, Shayan Chowdhury, and Fatih Uysal. SWE-Bench-CL: Continual learning for coding agents, 2025. URL https://arxiv.org/abs/2507.00014.

Michael Laskin, Luyu Wang, Junhyuk Oh, Emilio Parisotto, Stephen Spencer, Richie Steigerwald, D. J. Strouse, Steven Hansen, Angelos Filos, Ethan Brooks, et al. In-context reinforcement learning with algorithm distillation. In *International Conference on Learning Representations (ICLR)*, 2023.

Jonathan N. Lee, Annie Xie, Aldo Pacchiano, Yash Chandak, Chelsea Finn, Ofir Nachum, and Emma Brunskill. Supervised pretraining can learn in-context reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Xiaofeng Lin, Sirou Zhu, Yilei Chen, Mingyu Chen, Hejian Sang, Ioannis Paschalidis, Zhipeng Wang, Aldo Pacchiano, and Xuezhou Zhang. Scaling in-context online learning capability of LLMs via cross-episode meta-RL. *arXiv preprint arXiv:2602.04089*, 2026.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations (ICLR)*, 2024.

Zichen Liu, Anya Sims, Keyu Duan, Changyu Chen, Simon Yu, Xiangxin Zhou, Haotian Xu, Shaopan Xiong, Bo Liu, Chenmien Tan, Chuen Yang Beh, Weixun Wang, Hao Zhu, Weiyan Shi, Diyi Yang, Michael Shieh, Yee Whye Teh, Wee Sun Lee, and Min Lin. GEM: A gym for agentic LLMs, 2026. URL https://arxiv.org/abs/2510.01051.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *International Conference on Learning Representations (ICLR)*, 2018.

Alexander Nikulin, Vladislav Kurenkov, Ilya Zisman, Denis Tarasov, Viacheslav Sinii, and Sergey Kolesnikov. XLand-MiniGrid: Scalable meta-reinforcement learning environments in JAX. In *Intrinsically Motivated Open-ended Learning Workshop at NeurIPS*, 2023.

Open-Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, et al. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*, 2021.

OpenAI. GPT-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wolczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, Jakob Nicolaus Foerster, Jack Parker-Holder, and Tim Rocktäschel. BALROG: Benchmarking agentic LLM and VLM reasoning on games, 2025. URL https://arxiv.org/abs/2411.13543.

Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019.

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, 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 (NeurIPS)*, 2023.

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 (ICLR)*, 2021.

David Silver and Richard S. Sutton. Welcome to the era of experience. *Google DeepMind position paper / Designing an Intelligence (MIT Press)*, 2025.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *International Conference on Learning Representations (ICLR)*, 2025.

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 (TMLR)*, 2024.

Jane X. Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z. Leibo, Rémi Munos, Charles Blundell, Dharshan Kumaran, and Matthew Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *AAAI Conference on Artificial Intelligence*, 2024.

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

Junhao Zheng, Chengming Shi, Xidi Cai, Qiuke Li, Duzhen Zhang, Chenxing Li, Dong Yu, and Qianli Ma. Lifelong learning of large language model based agents: A roadmap. *arXiv preprint arXiv:2501.07278*, 2025b.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*, 2024.

Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. VariBAD: A very good method for bayes-adaptive deep rl via meta-learning. In *International Conference on Learning Representations (ICLR)*, 2020.

A Environment Details

We use seven text-game environments, each instantiated as a multi-task sequence by sampling a latent $z \sim p(z)$ once per sequence and then sampling task-specific variation ξ_i for each task τ_i^z in the sequence. For every environment we list (i) the single-task game; (ii) the multi-task / sequenceing version; (iii) the full latent catalogue, and the prompt template shown to the agent at task start. Reward, observation, and action space conventions are identical across latents within an environment.

Environment	Core task	Representative latent structures
Number Guessing	Identify hidden number	Restricted target set, range pattern
Bandits	Select rewarding arm	Shared best arm, reward pattern
Secretary	Accept/reject candidates	Threshold, position pattern
Mastermind	Infer hidden code	Code constraint, structural rule
Word Ladder	Transform word through graph	Hub word, reusable path structure
Wordle	Guess hidden word from feedback	Shared word property, candidate-set
Hangman	Reveal word through letter guesses	Shared word category, letter pattern

Table 6. Environment suite instantiated from the framework. Each environment defines a within-task decision problem and a family of shared latent structures.

All seven environments are declared the same way. Figure 5 shows the complete definition of number guessing (left) and how a configuration is composed from the registered components by name (right).

Define an environment

```python
from latentgym.core.registry import register_env
from .core_env import NumberGuessingSingleEpisodeEnv
register_env(
    "number_guessing",
    NumberGuessingSingleEpisodeEnv,
    min_range=1, max_range=1000,
    max_turns_per_episode=30)
```

Compose a configuration

```python
FullyDefinedEnv(
    "number_guessing", # dynamics
    "set_of_3",        # latent
    "no_info",         # prompt
    "standard",        # feedback
    num_episodes=10,   # horizon
)
```

Figure 5. Define once, compose many. An environment registers its core dynamics (left, number guessing in seven lines); its latents, prompts and feedbacks register separately, and any combination composes into a runnable environment (right). Swapping `set_of_3` for `set_of_2` changes only latent-identification difficulty.

A.1 Number Guessing

Single-task game. In this game the agent's task is to identify a hidden integer $y \in [\ell, u]$ that the environment has chosen uniformly at random. On each turn the agent submits a guess $g_t \in [\ell, u]$ and is told whether the hidden number is greater than, less than, or equal to the guess. This comparison is the only intermediate signal the agent receives. The episode ends as soon as the agent guesses correctly; otherwise it terminates after $T_{\max} = 30$ turns.

Reward. The per-task reward is decay-shaped so that the agent is incentivised to solve the puzzle quickly. If the agent guesses correctly on turn t, it receives

$$R \;=\; \max\bigl(0,\; 1 - 0.020 \cdot t\bigr),$$

and zero if it never guesses correctly within the turn cap. With $T_{\max} = 30$ and the per-turn decay of 0.020, the reward is at least 0.40 for any solution found within the cap and approaches 1.0 as $t \to 1$.

Multi-task version. In the multi-task version the agent plays a sequence of $N = 10$ number guessing tasks back-to-back. A latent variable z, sampled once at the beginning of the sequence and held fixed throughout, governs how the targets $y_1, \ldots, y_N$ are correlated across tasks (for instance, all targets may be drawn from the same fixed three-element set). The agent's context carries the full cross-task history $h_{<i}$, every guess–feedback pair from the preceding tasks together with the inter-task transition messages, so an attentive agent can in principle infer z within a few tasks and exploit it on the rest.

Latents. We use four latents in our experiments:

- `set_of_2` — all targets in the sequence are drawn from the same set of two specific numbers in $[1, 1000]$.
- `set_of_3` — all targets in the sequence are drawn from the same set of three specific numbers in $[1, 1000]$.
- `range_100` — all targets fall within a fixed contiguous 100-number sub-range of $[1, 1000]$.
- `dynamic_range` — targets fall within a contiguous 1000-number sub-range of a dynamic outer range that shifts deterministically across tasks.

Additional latents. For completeness, the codebase also registers two further latents that we do not evaluate in this paper: `range_1000` (targets fall within a fixed contiguous 1000-number range of $[1, 10000]$) and `dynamic_full_range` (the full $[\ell, u]$ prompt range itself slides as $[n, n+1000]$ where n varies across tasks in the sequence). We retain these in the framework so that future work can study cross-task adaptation when the prompt-visible range itself becomes part of the latent.

Environment Prompt (no info)

You are playing 10 number guessing games sequentially. In each game, you need to guess an integer (whole number, not a decimal) between 1 and 1000 (inclusive).

For each guess, I will tell you if the number is greater than, less than, or equal to your guess. Your goal is to guess the number correctly in as few turns as possible.

Your guess must be wrapped in square brackets. Format your guess as: `[number]` (e.g., `[500]`).

some_info adds: The numbers in these games might follow a pattern. Pay attention to the numbers you encounter across different games, as this might help you improve your strategy.
full_info adds (set_of_3 example): IMPORTANT: All the numbers in this series of games are drawn from a set of 3 specific numbers. Use the information from previous games to identify this set and improve your strategy.

A.2 A HISTORY-BLIND PERFORMANCE BOUND FOR NUMBER GUESSING

Consider a target sampled uniformly from $\{1, \ldots, 1000\}$, with greater-than, less-than, or equal feedback and reward $r(t) = 1 - 0.02t$ for a correct guess on turn t, with a 30-turn cap and zero reward on failure. A history-blind policy may use all feedback within the current task but receives no information from preceding tasks. Its comparison tree can identify at most 2^{t-1} targets for the first time on turn t. Filling the earliest levels maximizes expected reward: 511 targets occupy levels 1–9, and the remaining 489 occupy level 10. A balanced search tree attains this bound and solves every target within the turn cap:

$$\mathbb{E}[t]_{\min} = \frac{\sum_{t=1}^{9} t\, 2^{t-1} + 10 \cdot 489}{1000} = 8.987, \qquad \mathbb{E}[r]_{\max} = 1 - 0.02(8.987) = 0.82026.$$

In the evaluated `set_of_2` setting, each sequence's hidden pair is sampled uniformly and independently of training, and each task selects uniformly from that pair, so the target remains marginally uniform over the full range. The bound therefore applies to any policy without previous-task information, even if it knows that a shared pair exists.

Earlier tasks can reveal the pair and permit higher rewards. The reported final-task means on `set_of_2`, 0.87 ± 0.01 for Qwen3-8B and 0.85 ± 0.01 for Gemma3-12B (Tables 17 and 18), exceed this history-blind expectation; uncertainties are standard errors across five decoding seeds. This is not performance above the optimum for a history-aware policy: knowing the pair permits an expected reward of 0.97.

A.3 BANDITS

Single-task game. In this game the agent's task is to identify the highest-reward arm out of $K = 5$ buttons (named `red`, `blue`, `green`, `yellow`, `purple`). Each button has a hidden Bernoulli

reward probability $p_a \in [0, 1]$ that is fixed within a task. On each turn the agent can either **explore** a button, pulling it and observing a binary reward $r_t \sim \mathrm{Bern}(p_{a_t})$, or **commit** by selecting a button as its final answer. The intermediate signal the agent receives is the per-turn binary reward from each pull. The episode ends as soon as the agent commits, or after $T_{\max} = 30$ turns, in which case the last explored button is treated as the commitment.

Reward. The per-task reward depends only on the correctness of the commitment, with a per-turn decay that incentivises early commitment. If the committed button is the true highest-probability arm at the time of commitment t, the agent receives

$$R \;=\; \max\bigl(0,\; 1 - 0.015 \cdot t\bigr),$$

and zero otherwise. The intermediate Bernoulli rewards observed during exploration are informative but do not directly contribute to the per-task score.

Multi-task version. In the multi-task version the agent plays a sequence of $N = 10$ bandit tasks back-to-back. The latent z, sampled once per sequence, governs how the arm-reward vector $(p_0, \ldots, p_4)$ is correlated across tasks (for instance, the same arm may always be the best, or the best arm may rotate through the indices in a predictable order). The agent's context retains every prior pull-and-reward pair across tasks, so an attentive agent can in principle infer z from a few tasks and commit much earlier on the rest.

Latents. We use four latents in our experiments:

- `loyal_favorite_0` — a single fixed arm is the highest-reward arm in every task of the sequence.
- `top_two_fixed` — the same pair of arms always occupies the top two positions; the agent only needs to explore between them.
- `even_indices_only` — the best arm is always at an even index (0, 2, or 4).
- `ping_pong` — the best arm oscillates between the first and last index across tasks ($0 \to 4 \to 0 \to 4 \ldots$).

Additional latents. For completeness, the codebase registers a much larger family of bandit latents that we do not evaluate in this paper. These include position-fixed variants (`fixed_probabilities`, `one_hot`, `bottom_excluded`, `odd_indices_only`); rotation/cycle variants (`clockwise_rotation`, `counterclockwise_rotation`, `swap_top_two`, `cycle_length_5`); binary-switch variants (`binary_switch_0_1`, `binary_switch_0_last`); cross-episode memory variants (`shadow`, `cold_hand`, `hot_hand`, `random_walk`, `mirror_mode`); mathematical-sequence variants (`skip_2`, `skip_3`, `fibonacci`, `prime_indices`, `triangular`); and the `same_ranking` variant in which the ordering of arms is fixed across the sequence while their exact probabilities vary. We retain these in the framework so that future work can study cross-task adaptation in richer bandit settings.

Environment Prompt (no info)

You are playing a Multi-Armed Bandit game with 5 buttons (red, blue, green, yellow, purple). Each button has a hidden probability of giving a reward (0 or 1). You have up to 30 turns.

On each turn you can either:

- Explore: `[button_name]` (e.g., `[red]`) — press a button and observe the reward.
- Select: `[select button_name]` (e.g., `[select red]`) — lock in your final answer.

Once you select, the game ends immediately. Your goal is to identify the button with the highest probability. Selecting correctly earlier gives a higher reward. If you don't select by the last turn, your last explore action counts as your selection.

IMPORTANT: Only use square brackets `[...]` to submit your action for this turn (either `[button_name]` to explore or `[select button_name]` to finalize). Do not use square brackets anywhere else in your response.

You will play 10 rounds of this game sequentially.

some_info adds: The reward probabilities across these rounds may share a hidden pattern or constraint. Pay attention to which buttons perform best across games.
full_info adds (loyal_favorite_0 example): IMPORTANT: There is a consistent pattern of buttons across games. Pay attention to which button wins.

A.4 Secretary Problem

Single-task game. In this game the agent's task is to pick the best of $n = 20$ candidates that arrive one at a time. Each candidate t is revealed with a numeric score $s_t \in [0, 1]$, and after seeing it the agent must irrevocably either accept the candidate (ending the episode immediately) or continue to the next one. Rejected candidates cannot be revisited; if the agent reaches the end of the sequence without having accepted, it is forced to accept the last candidate. The intermediate signal the agent receives during a task is exactly the running sequence of revealed scores $s_1, s_2, \ldots$; there is no comparison feedback or hint about the underlying score distribution.

Reward. The per-task reward depends on the score of the accepted candidate relative to the true maximum of the full sequence $s_{\max} = \max_t s_t$. If the agent accepts the maximum, it receives the full reward of 1.0; otherwise it receives a partial credit proportional to how close the accepted score is to the maximum:

$$R = \begin{cases} 1.0 & \text{if } s_{\text{accepted}} = s_{\max}, \\ 0.5 \cdot \dfrac{s_{\text{accepted}}}{s_{\max}} & \text{otherwise.} \end{cases}$$

A random-stopping baseline averages roughly $R \approx 0.26$ under this scheme; an agent that always commits on a near-best candidate ($s_{\text{accepted}}/s_{\max} \approx 0.95$) earns $R \approx 0.47$ when it is wrong, well below the 1.0 available for picking the true maximum.

Multi-task version. In the multi-task version the agent plays a sequence of $N = 10$ secretary tasks back-to-back. The latent z, sampled once per sequence, governs how the position or score of the winning candidate is correlated across tasks (for instance, the winner may always appear at a fixed position, or its score may always exceed a hidden threshold). The agent's context retains the full score-sequence-and-decision history from preceding tasks, so an attentive agent can in principle infer z within a few tasks and exploit it on the rest.

Latents. We use four latents in our experiments:

- `inverse_order` — candidate scores arrive in strictly decreasing order, so the first candidate is always the best (decreasing).
- `fixed_position_2` — the winning candidate always appears at position 2 of the sequence.
- `sorted_order` — candidate scores arrive in strictly increasing order; the optimal policy is to always continue to the last candidate.
- `best_is_last` — the best candidate is always the last one presented (used as the OOD held-out latent).

Additional latents. For completeness, the codebase registers a much larger family of secretary latents that we do not evaluate in this paper, covering position-fixed, threshold, position-shift, cycle, structural, relational, cross-episode memory, and mathematical-position families. We retain these in the framework so that future work can study cross-task adaptation in richer secretary settings.

Environment Prompt (no info)

You are playing the Secretary Problem (optimal stopping). You will see 20 values one at a time. For each value, decide: `[accept]` to claim it, or `[continue]` to skip and see the next. Once you accept, the game ends. If you reach the last value, you must accept it. You win (reward 1.0) if the

value you accept is the maximum of all values. Otherwise you get partial credit based on how close your accepted value is to the maximum.

IMPORTANT: Output your action as `[accept]` or `[continue]`. Square brackets must be used ONLY for the final action — never elsewhere in your response.

You will play 10 rounds of this game sequentially.

some_info adds: The value sequences may have a pattern in where the maximum appears. Pay attention to previous rounds to improve your strategy.
full_info adds (threshold_06 example): IMPORTANT: Across games there is a consistent pattern. There is a value threshold — the maximum is always above it and other values are below. Discover the threshold to accept immediately.

A.5 MASTERMIND

Single-task game. In this game the agent's task is to crack a hidden 4-digit code, where each digit is independently drawn from $\{1, \ldots, 6\}$ (repeats allowed). On each turn the agent submits a length-4 guess and receives the standard Mastermind feedback as the intermediate signal: a count of *black pegs* (digits that are correct in both value and position) and a count of *white pegs* (digits that are correct in value but in the wrong position). The episode ends as soon as the agent submits the exact code (all four black pegs) or after $T_{\max} = 10$ guesses.

Reward. The per-task reward measures how close the final guess was to the hidden code, computed as the fraction of pegs that were correct in both value and position at the time the episode terminated:

$$R = \frac{\text{black pegs on final guess}}{4}.$$

A correct solve gives $R = 1.0$; a complete miss gives $R = 0.0$. Crucially, this reward depends only on the *final* guess and not on the trajectory of intermediate guesses, so the agent is rewarded for converging to the correct code rather than for incremental progress along the way.

Multi-task version. In the multi-task version the agent plays a sequence of $N = 10$ Mastermind tasks back-to-back. The latent z, sampled once per sequence, constrains the family of codes that appear across the sequence, for instance, all codes might be strictly ascending sequences, or all might begin with a fixed digit. The agent's context retains the full guess-and-feedback history from preceding tasks, so an attentive agent can in principle infer z within a few tasks and use it to drastically narrow the search space on the remaining ones.

Latents. We use four latents in our experiments:

- `strictly_ascending` — every hidden code is a strictly ascending sequence of four distinct digits.
- `strictly_descending` — every hidden code is a strictly descending sequence of four distinct digits.
- `first_is_6` — the first digit of every hidden code is fixed to 6.
- `has_pair` — every hidden code contains at least one repeated digit (used as the OOD held-out latent).

Additional latents. For completeness, the codebase registers a much larger family of mastermind latents that we do not evaluate in this paper, covering weakly monotonic patterns, palindromic and alternating structures, parity constraints, digit-set constraints, repeat / no-repeat constraints, and adjacency rules. We retain these in the framework so that future work can study cross-task adaptation under richer code constraints.

Environment Prompt (no info)

You are playing Mastermind. A secret code of 4 digits has been chosen, each digit between 1 and 6. Each turn, guess the code using format `[1 2 3 4]`. You'll receive feedback: black pegs (correct digit in correct position) and white pegs (correct digit in wrong position). Win by finding the exact code.

IMPORTANT: Only use square brackets `[X X X X]` for your actual guess submission. Do not use square brackets when discussing or referencing previous guesses.

You will play 10 rounds of Mastermind sequentially.

some_info adds: The secret codes may share a common pattern. Pay attention to the codes you discover across games.
full_info adds (strictly_ascending example): All secret codes share a hidden structural pattern (e.g., ascending order, palindrome, specific digit constraint). Discover this pattern to crack codes faster in later rounds.

A.6 WORD LADDER

Single-task game. In this game the agent's task is to transform a given start word into a given target word of the same length by changing exactly one letter at a time, where every intermediate word in the chain must itself be a valid English word from the supplied dictionary. On each turn the agent submits the next word in the chain; the intermediate signal it receives is whether the proposed word is accepted into the chain (i.e. differs from the previous word by exactly one letter and is in the dictionary) along with the updated current word. The episode ends as soon as the agent reaches the target word or after $T_{\max} = 20$ turns.

Reward. The per-task reward distinguishes solved from unsolved tasks. If the agent reaches the target word, it receives a decay-shaped reward that penalises taking more turns than the optimal solution:

$$r \;=\; \max\bigl(0,\; 1 - 0.03 \cdot (t - t^*)\bigr),$$

where t is the number of turns used and t^* is the length of the shortest valid chain from the start word to the target. A solution that matches the optimal length earns $R = 1.0$; each extra turn beyond optimal costs 0.03, so for example a 3-turn overshoot earns $R = 0.91$ and a 10-turn overshoot earns $R = 0.70$. If the agent fails to reach the target within the turn cap, the reward is the fraction of letter positions in the current final word that match the target:

$$r \;=\; \frac{\text{positions matching target}}{|\text{target}|}.$$

This partial credit gives a smooth signal even on failed attempts.

Multi-task version. In the multi-task version the agent plays a sequence of $N = 10$ Word Ladder tasks back-to-back. The latent z, sampled once per sequence, constrains either the family of valid intermediate words or the structure of the optimal solution path (for instance, every optimal chain may pass through a fixed pivot word, or every chain may only substitute consonants). The agent's context retains the full solution-chain history from preceding tasks, so an attentive agent can in principle infer z within a few tasks and use it to shortcut the search on the rest.

Latents. We use four latents in our experiments:

- `hub_word_3letter` — every (start, target) pair in the sequence admits a shortest-path solution that passes through a fixed 3-letter "hub" word. Discovering the hub splits each puzzle into two easier sub-problems.
- `hub_word_4letter` — same as above, with a fixed 4-letter hub word.
- `order_outside_in` — at least one optimal path changes outer letter positions first and inner positions last.
- `order_left_to_right` — at least one optimal path changes letter positions strictly left-to-right (used as the OOD held-out latent).

Additional latents. For completeness, the codebase registers a much larger family of word-ladder latents that we do not evaluate in this paper, covering restricted-vocabulary paths, alternative position-order strategies (right-to-left), substitution-type constraints (vowel-only, consonant-only, alternating, phonetic-group), word-family constraints (paths that stay within the words containing a given substring or sharing a fixed suffix/prefix), and skeleton constraints on the start/target pair. We retain these in the framework so that future work can study cross-task adaptation under richer Word Ladder structures.

Environment Prompt (no info)

You are playing a series of 10 Word Ladder games. In each game, you are given a start word and a target word. Transform the start word into the target word by changing exactly one letter at a time. Each intermediate word must be a valid English word.

You may reason about your strategy, but your answer must be wrapped in square brackets. Format your guess as: `[word]` (e.g., `[cold]`).

some_info adds: The word pairs and their solutions across games may share common properties or patterns. Pay attention to the words that appear in your solutions — this might help you solve future games faster.
full_info adds (hub_word_4letter example): IMPORTANT: There is a pattern across games. All the word pairs in this series can be solved by going through a specific intermediate 'hub' word. Discover this hub word and use it to split every puzzle into two easier sub-problems.

A.7 WORDLE

Single-task game. In this game the agent's task is to identify a hidden 5-letter English word that the environment has chosen from a fixed dictionary. On each turn the agent submits a 5-letter guess; the intermediate signal it receives is the standard Wordle feedback, returned per-letter as one of three colours: green (the letter is correct in both identity and position), yellow (the letter appears in the target but at a different position), or grey (the letter does not appear in the target at all). The episode ends as soon as the agent submits the exact target word or after $T_{\max} = 6$ guesses.

Reward. The per-task reward is binary at the trajectory level and depends on whether the agent identified the target within the turn cap. If the agent guesses the target exactly, it receives the full reward of 1.0; otherwise it receives a partial credit equal to the fraction of letters that ended up correctly identified at the correct position by the final guess (the per-letter *percentage completion* returned by the underlying TextArena environment). A complete miss yields $R = 0$.

Multi-task version. In the multi-task version the agent plays a sequence of $N = 10$ Wordle puzzles back-to-back. The latent z, sampled once per sequence, constrains the family of target words used across the sequence, for instance, every target may use only high-frequency English letters, or every target may avoid repeated letters. The agent's context retains the full guess-and-feedback history from preceding tasks, so an attentive agent can in principle infer z within a few tasks and use it to seed strong opening guesses on the rest.

Latents. We use four latents in our experiments:

- `vowel_count_3` — every target word contains exactly three vowels.
- `vowels_clustered` — every target word contains at least two adjacent vowels.
- `position_E_at_1` — the second letter of every target word is `E`.
- `has_double_letter` — every target word contains a pair of consecutive identical letters (used as the OOD held-out latent).

Additional latents (designed but not used). For completeness, the codebase registers a much larger family of Wordle latents that we do not evaluate in this paper, covering vowel-count and vowel-position constraints, position-fixed letter constraints, alphabet-half restrictions, consonant-vs-vowel pattern templates (CVCVC, CVCCV, VCCVC), specific-letter "contains" / "starts-with" /

"ends-with" patterns, double-letter constraints, and semantic-category restrictions. We retain these in the framework so that future work can study cross-task adaptation under richer Wordle target distributions.

Environment Prompt (no info)

You are playing Wordle. A secret 5-letter word has been chosen. You have 6 attempts to guess it.

For each guess, wrap your word in square brackets (e.g., `[apple]`). Feedback for each letter:

- G (green): correct letter in the correct position.
- Y (yellow): letter exists in the word but in the wrong position.
- X (wrong): letter is not in the word.

IMPORTANT: Output your guess as `[word]`. Square brackets must be used ONLY for the final guess — never elsewhere in your response.

You will play 10 rounds of this game sequentially.

some_info adds: The secret words across rounds may share a hidden pattern. Pay attention to common features of the target words.
full_info adds (letter_freq_high example): IMPORTANT: Average letter frequency $> 7\%$. Use this knowledge to guess more efficiently in later rounds.

A.8 HANGMAN

Single-task game. In this game the agent's task is to identify a hidden English word that the environment has chosen from a fixed dictionary. On each turn the agent submits either a single letter or a complete word guess. The intermediate signal is the partially-revealed board: a guessed letter that appears in the target is unmasked at every matching position, while a guess that is not in the target costs the agent one of its $T_{\max} = 6$ allowed wrong guesses. A complete word guess wins immediately if correct or burns one wrong attempt if not. The episode ends as soon as the entire word is revealed, the agent submits the correct word, or the wrong-guess budget reaches zero.

Reward. The per-task reward is binary on a clean win and a fraction of revealed letters otherwise. If the agent reveals the entire word (letter-by-letter or via a correct full-word guess), it receives the full reward of 1.0. If it exhausts the wrong-guess budget, it receives partial credit equal to the fraction of unique target letters that had been correctly revealed by the time the episode ended,

$$r = \frac{\text{distinct target letters revealed}}{\text{total distinct letters in target}}.$$

This gives a smooth signal even on failed attempts and rewards the agent for narrowing down the target before running out of guesses.

Multi-task version. In the multi-task version the agent plays a sequence of $N = 10$ Hangman tasks back-to-back. The latent z, sampled once per sequence, constrains the family of hidden words that appear across the sequence, for instance, every target may share a suffix, a vowel count, or a semantic category. The agent's context retains the full guess-and-board history from preceding tasks, so an attentive agent can in principle infer z within a few tasks and use it to seed strong opening guesses on the rest.

Latents. We use four latents in our experiments:

- `vowel_count_4` — every target word contains exactly four vowels.
- `ending_ABLE` — every target word ends in the suffix "ABLE".
- `has_double_letter` — every target word contains a pair of consecutive identical letters (e.g. `LL`, `EE`, `SS`).
- `starts_with_S` — every target word begins with the letter `S`.

Additional latents (designed but not used). For completeness, the codebase registers a much larger family of Hangman latents that we do not evaluate in this paper, covering word-length constraints, alternate vowel counts and vowel/consonant ratios, semantic categories (e.g. "physical objects"), letter- frequency-score classes, "contains-letter" / "avoids-letter" constraints, alternate suffix and consonant-cluster patterns, and overall difficulty bands (easy/medium/hard). We retain these in the framework so that future work can study cross-task adaptation under richer Hangman target distributions.

Environment Prompt (no info)

You are playing Hangman. A secret word has been chosen. Each turn, guess a letter using `[L]` format (e.g., `[A]`) or guess the full word using `[WORD]` format (e.g., `[APPLE]`). Correct letters are revealed on the board. Wrong guesses cost an attempt. Win by revealing all letters or guessing the word before running out of attempts.

IMPORTANT: Only use square brackets `[...]` to submit your actual guess. Do not use square brackets anywhere else in your response.

You will play 10 rounds of Hangman sequentially.

some_info adds: The words may share common properties. Pay attention to patterns across games to improve your guessing strategy.
full_info adds (ending_ABLE example): All target words share a hidden property (e.g., same length, same starting letter, same vowel pattern). Discover this property from early rounds to guess more efficiently in later rounds.

A.9 Prompt types

Prompt type	Information revealed	Purpose
No information	No indication that tasks share a latent structure	Tests whether the agent discovers structure from experience alone
Vague hint	States that some recurring structure may exist	Tests whether the agent identifies the cross-task regularities efficiently
Full information	Describes the latent; in some environments only its family	Tests whether the agent can use latent information once given

Table 7: Prompts vary how much information about the shared latent is available before interaction.

A.10 Feedback types (orthogonal to prompt).

After every task in the sequence the agent receives one of:

- `standard` — only the per-task reward signal (success / fraction-correct), identical to what the agent already observes during the task.
- `information` — the per-task reward *plus* the ground-truth target / code / winning candidate from the just-finished task. This lets the agent verify or revise its hypothesis about the latent z.

The feedback string is appended to the cross-task context $h_{<i}$ before task $i{+}1$ begins; the prompt condition (no / some / full info) remains unchanged across tasks within a sequence.

A.11 A CONCRETE COMPARISON WITH CONTINUAL-LEARNING BENCHMARKS

In SWE-Bench-CL (Joshi et al., 2025), a coding agent solves a sequence of issues in the same software project, creating opportunities to reuse what it learns. Consider two actual tasks involving Django, software that translates Python database operations into SQL queries. The boxes below explain each problem and its actual fix, using illustrative numbers to clarify what the software should do.

Earlier task: count unique values

Goal. Count unique values among records satisfying a condition. If the selected values are $[2, 2, 5]$, the answer should be 2.

What goes wrong? Django joins two SQL keywords without a space, producing an invalid database query.

Actual fix. Insert the missing space:
`DISTINCTCASE WHEN`
$\longrightarrow$ `DISTINCT CASE WHEN`

Task ID: `django__django-10880`

Later task: sum or average them

Goal. Sum or average unique values. For $[2, 2, 5]$, the sum should be 7 and the average 3.5, using only $[2, 5]$.

What goes wrong? Django rejects the request because its sum and average operations do not allow the distinct-value option.

Actual fix. Enable that option in both classes, `Sum` and `Avg`:
`allow_distinct = True`

Task ID: `django__django-11603`

What links these tasks?

Both fixes change the same file, `django/db/models/aggregates.py`, which implements database calculations such as counts and sums. The dataset therefore links the later task to the earlier one. Solving the first task could teach the agent where to look and how Django handles distinct values. But the second task requires a different change: enabling an option rather than repairing SQL spacing. The link identifies a possible opportunity for reuse, without specifying exactly what knowledge should transfer or how much it should help.

What remains ambiguous? The dataset provides correct patches and tests, but does not designate a particular shared rule as the ground-truth latent or control the benefit of knowing that rule. If the agent fails on the later task, several explanations remain possible:

1. The earlier fix offered little knowledge useful for the later task.
2. Useful knowledge was available, but the agent did not acquire or retain it.
3. The agent retained useful knowledge but failed to apply it correctly.

Task success alone does not distinguish these explanations. SWE-Bench-CL supports measuring transfer; our comparison concerns isolating what transfers and how it is used. In complex coding tasks, identifying and varying that shared structure independently of the individual problems is difficult.

A controlled counterpart in LATENTGYM

Known shared structure: targets lie in $[1, 1000]$ but are drawn from the fixed hidden set $z = \{137, 793\}$. An illustrative sequence is:

Task 1: 137 $\longrightarrow$ Task 2: 793 $\longrightarrow$ Task 3: 137

Why it can help. An agent that knows the set can identify any target in at most two guesses, instead of searching the full range.

What we can test. Compare hidden-latent and revealed-latent prompts on the same task sequence to measure the benefit of providing this information. Vary the hidden-set size at a fixed visible range to change how informative the latent is.

Here the experimenter knows exactly which information is shared and can intervene on its availability. This supports controlled studies of discovering and using shared structure without requiring the same structure to be identified in a real codebase. The comparison illustrates experimental design, rather than an evaluation of LATENTGYM on coding tasks.

B Experimental Details

Table reporting conventions. Unless stated otherwise, every result table reports the mean ± standard error across five independent decoding seeds, with 50 trajectories per latent and seed. In-distribution results average over the training latents of the evaluated model (three per environment for single-environment models, two for the multi-environment model); OOD-1 results use the held-out latent of each environment, and OOD-2 results the two multi-environment latents. Agent-switching tables pool all four latents of their environment, except the equal-budget switching table, which uses `set_of_3` alone; the per-latent breakdowns report each latent separately. The training-seed study (Table 23) additionally reports the mean ± standard error across three independent training seeds.

Base model and infrastructure. We use `Qwen3-8B` and `Gemma3-12B` as our base model for all RL fine-tunes. Our training pipeline is built on top of SkyRL, with rollouts generated through a colocated `vLLM` inference server. Each run uses 4× NVIDIA A100 80GB GPUs. We optimise the policy with GRPO, using the standard clipped objective and a KL penalty against a frozen reference policy π_{ref}.

Training-time prompt and feedback condition. Unless stated otherwise, all RL fine-tunes in this paper are run under the `some_info` prompt condition (vague hint that a cross-task pattern may exist across tasks) and the `standard` feedback condition (the agent receives only the per-task reward between tasks).

Shared hyperparameters across all RL runs. We tune the GRPO hyperparameters once on the Number Guessing environment and then fix them for every other experiment in this paper. Final values are listed in Table 8. The only across-experiment variation is the sequence length per rollout (N=10 tasks for cross-task runs vs. N=1 for single-task runs).

Table 8: Shared GRPO hyperparameters across all RL fine-tunes.

Hyperparameter	Qwen3-8B	Gemma3-12B
Optimizer	AdamW (β_1=0.9, β_2=0.999)	AdamW (β_1=0.9, β_2=0.999)
Policy learning rate	5×10^{-7}, 3×10^{-7}	2×10^{-7}, 4×10^{-7}
Weight decay	0.01	0.01
KL coefficient β	0.04	0.2, 0.1
Clip range ϵ	0.2	0.2
Group size (rollouts per prompt)	8	8
Train batch size (`tbs`)	32 prompts	32 prompts
Mini-batch size (`mbs`)	16	16
Samples per prompt (`nsp`)	8	8
Max response tokens	256	256
sequence length per rollout	N=10 tasks (`ne=10`) or N=1 task (`ne=1`)	
Reward functional Φ	$\sum_i G_i$ (cumulative return)	$\sum_i G_i$ (cumulative return)
Reference policy	frozen Qwen3-8B	frozen Gemma3-12B-IT
Sampling temperature (training)	0.8	1.0
Sampling top-p (training)	0.95	0.95
Sampling temperature (eval)	0.70	0.70
Sampling top-p (eval)	1.0	1.0
Random seed	263 ([263, 42, 1729] if multiple)	263

B.1 Single-environment Experiments

Training. For each of the six core environments (Number Guessing, Mastermind, Hangman, Word Ladder, Secretary, Wordle) we train two variants from the shared base model: a cross-task model (`ne=10`) and a single-task model (`ne=1`). Both variants train on the three in-distribution latents listed below for each environment.

- Number Guessing — `set_of_3`, `set_of_2`, `dynamic_range`.
- Mastermind — `strictly_ascending`, `strictly_descending`, `first_is_6`.

- Hangman — `vowel_count_4`, `has_double_letter`, `starts_with_S`.
- Word Ladder — `hub_word_4letter`, `hub_word_3letter`, `order_outside_in`.
- Secretary — `inverse_order`, `fixed_position_2`, `sorted_order`.
- Wordle — `vowel_count_3`, `vowels_clustered`, `position_E_at_1`.

Each training step samples a batch of 32 prompts and generates 8 rollouts per prompt, where one rollout is a complete sequence of N tasks. At every step the latent of each prompt is uniformly sampled from the three training latents. Training proceeds to saturation, i.e. until reward plateaus on held-out validation; we checkpoint periodically and select the best checkpoint by validation cumulative return on a held-out seed.

Evaluation. We evaluate each fine-tuned variant, and the base models (Qwen3-8B, Gemma3-12B) for comparison, under the same conditions used during training (`some_info` prompt and `standard` feedback). For every environment we evaluate on four latents: the three in-distribution latents seen during training, together with one held-out (OOD-1) latent. The held-out latents are:

- Number Guessing — `range_100`.
- Mastermind — `has_pair`.
- Hangman — `ending_ABLE`.
- Word Ladder — `order_left_to_right`.
- Secretary — `best_is_last`.
- Wordle — `has_double_letter`.

We run 50 trajectories of task-sequences per (model, environment, latent) cell, for a total of 200 trajectories per environment for each model.

B.2 Multi-environment Experiments

Training. A single cross-task model and a single single-task model are trained on sequences sampled from the union of four core environments: Number Guessing, Hangman, Word Ladder, and Secretary. Each sequence's environment is held fixed across its $N = 10$ tasks (within-sequence variation is over the latent's task-specific ξ_i, not over the environment), and the sequence's latent is uniformly sampled from that environment's two in-distribution training latents listed below:

- Number Guessing — `set_of_3`, `range_100`.
- Hangman — `vowel_count_4`, `ending_ABLE`.
- Word Ladder — `hub_word_4letter`, `hub_word_3letter`.
- Secretary — `inverse_order`, `fixed_position_2`.

Total training latents: 4 envs $\times$ 2 latents $= 8$. Training proceeds to saturation with the same shared hyperparameters.

Evaluation. We evaluate the two multi-environment fine-tuned variants, and the base models (Qwen3-8B, Gemma3-12B) for comparison, with `some_info` prompt and `standard` feedback. Note that these are the same training conditions as the model trained on $N = 10$ while the single-task model is trained on `some_info` given that it does not access the previous history. Each model is evaluated on every one of the four core environments separately, on the two in-distribution training latents per environment listed above, with 50 trajectories of $N = 10$ tasks per (model, environment, latent) cell, for a total of 100 trajectories per environment.

B.3 Multi-environment Prompt and Feedback Sweep Experiments

Training. We use the same setup as the multi-environment experiments above (four core environments: Number Guessing, Hangman, Word Ladder, Secretary, with the same two in-distribution

training latents per environment listed in Section B.2, for a total of 4 envs $\times$ 2 latents $=$ 8 training latents). We train only cross-task models (`ne=10`); for each of the four (prompt, feedback) combinations

$$(\text{prompt}, \text{feedback}) \in \{\texttt{some_info}, \texttt{full_info}\} \times \{\texttt{standard}, \texttt{information}\},$$

we train three independent models with seeds $\{42, 263, 1729\}$, for a total of 4 combos $\times$ 3 seeds $=$ 12 models. Training proceeds to saturation with the same shared hyperparameters.

Evaluation. Each fine-tuned model, and the base Qwen3-8B for comparison, is evaluated on every one of the four prompt $\times$ feedback combinations (yielding both in-distribution and cross-condition cells), on every one of the four core environments, on the two in-distribution training latents per environment, with 50 trajectories of $N = 10$ tasks per (model, environment, latent, eval-prompt, eval-feedback) cell. This gives 4 envs $\times$ 2 latents $\times$ 4 eval combos $=$ 32 eval cells per model and a total of 1,600 trajectories per model.

B.4 Leave-one-out Experiments

Training. We use the same setup as the multi-environment experiments above, except that one environment e is held out at a time. For each held-out $e \in \{$NG, hangman, wordladder, secretary$\}$ we train one cross-task model on sequences sampled from the *other three* environments, using the same two in-distribution training latents per training environment listed in Section B.2 (total 3 envs$\times$ 2 latents $=$ 6 training latents per leave-one-out model). The held-out environment e itself is never seen at training time. Training proceeds to saturation with the same shared hyperparameters.

Evaluation. We use the same evaluation setup as the multi-environment experiments above (`some_info` prompt, `standard` feedback, 50 trajectories of $N = 10$ tasks per cell). The only change is that each leave-one-out model, and the base models (Qwen3-8B, Gemma3-12B) for comparison, is evaluated only on its held-out environment e, on the two in-distribution latents of that environment, for a total of 100 trajectories per held-out environment. This is the OOD-2 setting reported in the paper.

B.5 Experience Collection vs. Utilization Experiments

This experiment uses the trained models from the multi-environment experiments (Section B.2), namely the base model, the cross-task multi-env model, and the single-task multi-env model, without further fine-tuning.

Evaluation. We use the same evaluation setup as the multi-environment experiments (Section B.2; `some_info` prompt, `standard` feedback, all four latents of each environment, 50 trajectories of $N = 10$ tasks per cell). For each model pair $(A, B) \in \{$(base, single-task), (base, cross-task), (single-task, cross-task)$\}$ and each switch point $K \in \{2, 4, 6, 8\}$, we run two directional rollouts:

- $A \to B$: model A acts during episodes $1, \ldots, K$ and collects the trajectory; model B inherits the cross-task context $h_{<K}$ and acts during episodes $K{+}1, \ldots, 10$.
- $B \to A$: the reverse.

Each model pair, switch point, and direction is evaluated on every one of the four core environments separately. Solo baselines come from each model's standalone single-agent eval on the same grid. ExploreGain and ExploitGain (Section 2) pool the four latents of each environment; the equal-budget variant (Table 22) uses `set_of_3` alone.

Interpretation. These comparisons distinguish the roles of collecting and using experience. The usefulness of a collected history depends on both its information content and how that information is presented to the receiving agent. Similarly, performance after the switch reflects both use of the inherited history and subsequent task-solving ability, including further exploration. The gains should therefore be interpreted relative to the reference agent and switch point.

B.6 Compute (Training and Evaluation)

We report cumulative wall-clock compute for all Qwen3-8B and Gemma3-12B fine-tunes and post-training evaluations in A100 80GB GPU-hours.

Training (RL fine-tunes). Table 9 lists the RL fine-tunes and their cost.

	Qwen3-8B		Gemma3-12B	
Experiment	Runs	A100-h	Runs	A100-h
Hyperparameter tuning and preliminary fine-tunes	–	∼1,000	–	∼1,000
Per-environment — 6 environments × 2 variants (`ne=10` and `ne=1`)	12	∼1,960	11	∼2,330
4-environment — (`ne=10` and `ne=1`)	2	∼330	2	∼430
Leave-one-out (`ne=10`)	4	∼270	4	∼350
Prompt and feedback sweep (Section B.3) — 4 prompt × feedback combinations × 3 seeds, cross-task	12	∼2,300	–	–
Equal-budget single-task RL (Table 21) — budget-matched `ne=1` models	3	∼195	–	–
Training-seed study (Table 23) — two further seeds × {cross-task, single-task}	4	∼210	–	–
Training subtotal	37	∼6,265	17	∼4,110

Table 9: Training compute. RL fine-tunes and their cost in A100 80GB GPU-hours.

Evaluation (post-training). Each fine-tuned variant and the base models (Qwen3-8B, Gemma3-12B) are evaluated on their respective grids (50 trajectories per cell, $N = 10$ tasks per trajectory); Table 10 lists the cost.

	A100-h	
Evaluation	Qwen3-8B	Gemma3-12B
Per-environment evals — fine-tuned variants + base across 6 environments	∼230	∼210
Multi-env, leave-one-out, and switch-point sweeps	∼140	∼140
Evaluation subtotal	∼370	∼350

Table 10: Evaluation compute. Post-training evaluation cost in A100 80GB GPU-hours.

Total. Training and evaluation of all Qwen3-8B models in this paper consumed approximately ∼6,635 A100 80GB GPU-hours, and of all Gemma3-12B models approximately ∼4,460.

B.7 Frontier-Model Evaluation

Models. We evaluate three production frontier models accessed through OpenRouter. Table 11 lists the exact model version behind each OpenRouter identifier and the dates over which the reported trajectories were collected. GPT-4o and Gemini 2.5 Flash are queried at temperature 0.7;

OpenRouter identifier	Exact version	Query dates
`openai/gpt-4o`	`gpt-4o-2024-08-06`	Apr 1 – May 6, 2026
`anthropic/claude-sonnet-4.6`	`claude-sonnet-4.6` (build 2026-02-17)	Apr 8 – 17, 2026
`google/gemini-2.5-flash`	`gemini-2.5-flash` (GA stable, released 2025-06-17)	Apr 8 – May 6, 2026

Table 11: Frontier models: exact versions and query dates.

Claude Sonnet 4.6 is queried at temperature 1.0. All three use `max_tokens`= 4096 per response. Where the provider supports it, reasoning is requested through OpenRouter: Gemini 2.5 Flash with

`reasoning: {effort: "high"}` and Claude Sonnet 4.6 with a 4096-token reasoning budget; GPT-4o has no reasoning mode. Routing to OpenRouter lets us evaluate every model through a single unified API and avoids per-provider client differences.

Evaluation grid. Unlike the trained-model evaluations, the frontier-model evaluation exercises the full prompt $\times$ feedback grid in order to test each model under `no_info`, `some_info`, and `full_info` prompts and both `standard` and `information` feedback conditions. For every (model, environment) pair, we evaluate on 4 latents $\times$ 3 prompts $\times$ 2 feedbacks $\times$ 5 trajectories $=$ 120 trajectories per environment. The four eval latents are the four canonical latents per environment listed in Section A.

Failure-case selection. To surface failures, we filter the trajectories for each (model, environment) pair by low score using an environment-specific threshold and then inspect the errors made in each failing trajectory. The transcripts shown in Appendix D are representative examples chosen to illustrate each qualitative failure category.

C Additional Results

C.1 SOTA Models Additional Results

We highlight again that we propose LatentGym as a diagnostic testbed rather than a benchmark: the numbers in this section are meant to ask, on every environment, whether frontier models improve across tasks without a cue, whether a vague hint changes this, and whether revealing the latent helps.

Table 12 reports cumulative reward, gain and final reward for each model and environment under the three prompt conditions. Given the magnitude of the standard errors, most Gain values are not statistically significant; without a cue, only Gemini 2.5 Flash on Number Guessing and Bandits and GPT-4o on Secretary show a Gain more than two standard errors above zero (adaptation neglect). The errors are wide because the reward of Secretary and Bandits is close to binary within a task; on Word Ladder and, for Claude Sonnet 4.6, Wordle and Mastermind, reward is near its maximum from the first task, leaving no room for Gain. Revealing the latent raises reward mainly on Number Guessing and lowers it in several cells, for example GPT-4o on Secretary and Mastermind and Claude Sonnet 4.6 on Hangman (adaptation miscalibration).

Table 13 separates noticing the structure from using it, on the running example of the main text: Number Guessing with three recurring targets. Without a cue, no model ever opens a later task with a previously revealed target, and the number of guesses does not fall from the first task to the last (adaptation neglect). With the vague hint, identification and exploitation come apart: Claude Sonnet 4.6 identifies the set and solves the last task in under two guesses, whereas Gemini 2.5 Flash identifies it but still needs about seven, and GPT-4o opens with a recurring target on only a third of the later tasks (adaptation breakdown). With the latent revealed, all three models open later tasks with a recurring target.

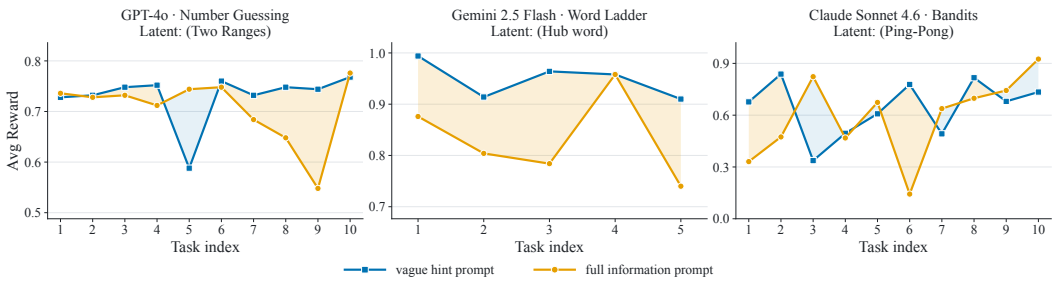


Figure 6. Examples where full information harms performance. Average reward across tasks under vague-hint and full-information prompts for GPT-4o on number guessing (two ranges), Gemini 2.5 Flash on Word Ladder (hub word), and Claude Sonnet 4.6 on bandits (ping-pong). The Word Ladder panel is also shown in the main text.

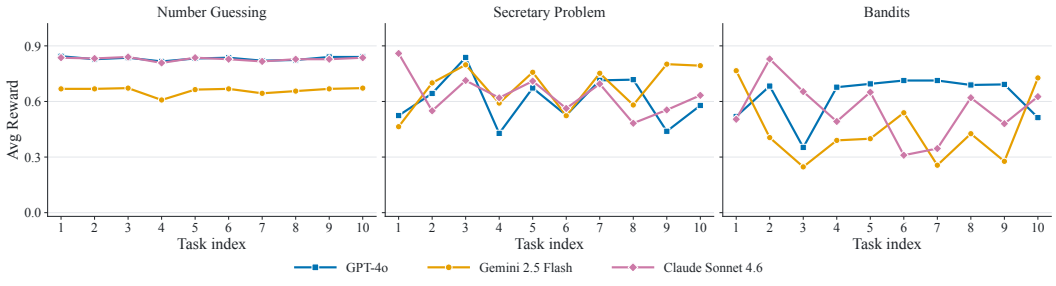


Figure 7. Average reward across tasks under the no-information prompt condition, for three frontier models (GPT-4o, Gemini 2.5 Flash, Claude Sonnet 4.6) on three environments: number guessing (set of 3), secretary (threshold 0.6), and bandits (best arm 0).

		No info			Some info			Full info		
	Env.	Cum.	Gain (%)	Final	Cum.	Gain (%)	Final	Cum.	Gain (%)	Final
Claude 4.6	Num. guess	7.9 ±0.03	+1.3 ±1.0	0.79 ±0.01	8.2 ±0.04	+6.4 ±1.7	0.84 ±0.01	**8.4** ±0.03	**+10.7** ±0.7	**0.86** ±0.00
	Mastermind	**9.6** ±0.07	**−6.8** ±2.6	**0.91** ±0.03	9.5 ±0.09	−6.9 ±2.5	0.92 ±0.03	9.5 ±0.12	−3.1 ±2.8	0.95 ±0.02
	Hangman	8.8 ±0.14	−1.0 ±5.3	0.89 ±0.05	**8.9** ±0.12	**+1.6** ±6.3	**0.94** ±0.03	8.2 ±0.06	+11.1 ±9.4	0.87 ±0.04
	Wordladder	**5.0** ±0.01	**+1.4** ±0.6	**0.99** ±0.00	**5.0** ±0.01	**+1.3** ±0.5	**1.00** ±0.00	**5.0** ±0.01	**+0.0** ±0.4	**1.00** ±0.00
	Secretary	7.0 ±0.33	+21.3 ±12.0	0.74 ±0.08	7.9 ±0.30	+25.6 ±25.3	0.80 ±0.07	**8.5** ±0.28	**+83.3** ±40.0	**0.82** ±0.05
	Wordle	**10.0** ±0.03	**0.0** ±0.0	**1.00** ±0.00	**10.0** ±0.02	**+0.5** ±0.5	**1.00** ±0.00	**10.0** ±0.01	**+1.0** ±0.6	**1.00** ±0.00
	Bandits	5.8 ±0.12	+16.8 ±21.1	0.63 ±0.03	6.5 ±0.22	+56.7 ±22.6	0.81 ±0.03	**6.8** ±0.19	**+59.4** ±34.7	**0.73** ±0.05
GPT-4o	Num. guess	7.9 ±0.03	+1.0 ±0.6	0.80 ±0.01	7.9 ±0.03	+1.5 ±1.1	0.80 ±0.01	**8.1** ±0.07	**+7.9** ±1.4	**0.85** ±0.01
	Mastermind	**7.0** ±0.13	**+2.9** ±13.4	**0.68** ±0.04	6.7 ±0.12	+6.3 ±11.9	0.71 ±0.06	5.9 ±0.16	−10.6 ±10.2	0.56 ±0.06
	Hangman	5.9 ±0.51	+14.1 ±20.2	0.62 ±0.09	4.3 ±0.82	+32.5 ±23.4	0.48 ±0.12	**6.9** ±0.39	**+2.4** ±12.7	**0.73** ±0.08
	Wordladder	4.3 ±0.08	+0.5 ±7.2	0.86 ±0.05	4.4 ±0.10	−1.5 ±5.3	0.89 ±0.04	**4.5** ±0.05	**+8.4** ±2.3	**0.92** ±0.02
	Secretary	**7.4** ±0.08	**+42.5** ±19.0	**0.83** ±0.04	7.2 ±0.36	+47.6 ±27.9	0.80 ±0.08	5.6 ±0.39	+45.0 ±36.6	0.60 ±0.08
	Wordle	7.5 ±0.22	−0.6 ±6.1	0.72 ±0.03	**7.8** ±0.11	**+14.4** ±5.5	**0.83** ±0.02	**7.8** ±0.12	**+3.7** ±7.4	**0.77** ±0.04
	Bandits	5.5 ±0.23	−9.3 ±14.2	0.49 ±0.06	5.7 ±0.19	+11.4 ±17.3	0.63 ±0.04	**5.9** ±0.25	**+49.6** ±58.1	**0.64** ±0.10
Gemini 2.5	Num. guess	7.4 ±0.23	+5.6 ±1.8	0.76 ±0.03	**7.8** ±0.20	**+9.1** ±3.5	**0.81** ±0.02	7.7 ±0.43	+11.5 ±1.9	0.80 ±0.05
	Wordladder	4.7 ±0.07	−8.1 ±2.9	0.90 ±0.04	**4.9** ±0.02	**−3.8** ±1.4	**0.96** ±0.02	4.7 ±0.10	−3.5 ±4.6	0.94 ±0.04
	Secretary	**3.0** ±0.70	**+29.2** ±28.8	**0.30** ±0.09	1.9 ±0.38	+156.8 ±133.9	0.24 ±0.04	2.2 ±0.38	+965.7 ±924.7	0.24 ±0.03
	Bandits	3.4 ±0.23	+41.2 ±13.6	0.40 ±0.01	4.2 ±0.31	+168.9 ±108.1	0.48 ±0.08	**4.8** ±0.15	**+377.5** ±199.5	**0.64** ±0.07

Table 12. Frontier models across environments under the three prompt conditions. Claude Sonnet 4.6, GPT-4o and Gemini 2.5 Flash on every environment. Mean ± SEM across five seeds; each seed pools the environment's 4 eval latents × 2 feedback conditions (8 trajectories), with N=10 tasks (N=5 for Word Ladder). *Cum.* is the total reward over the sequence ($R = \sum_i r_i$); *Gain* is the improvement from first to last task as a percentage of the initial reward ($(r_N - r_1)/r_1$); *Final* is the reward on the last task (r_N).

	No info			Some info			Full info		
Model	Use (%)	Turns_1	Turns_{10}	Use (%)	Turns_1	Turns_{10}	Use (%)	Turns_1	Turns_{10}
Claude Sonnet 4.6	0 ±0	8.2 ±0.9	8.2 ±0.9	100 ±0	7.8 ±0.7	1.8 ±0.4	100 ±0	8.4 ±0.9	1.6 ±0.2
GPT-4o	0 ±0	7.8 ±0.7	8.0 ±0.8	36 ±22	7.8 ±0.7	6.8 ±1.7	100 ±0	7.8 ±0.7	1.2 ±0.2
Gemini 2.5 Flash	0 ±0	12.6 ±4.4	12.4 ±4.5	96 ±4	10.8 ±1.8	6.8 ±2.4	87 ±13	16.2 ±5.3	7.0 ±5.8

Table 13. Do frontier models use the recurring structure? Number Guessing, `set_of_3`. Every target in a sequence is one of three hidden numbers. *Use* is the share of tasks 6–10 whose first guess is one of the sequence's recurring targets; *Turns*$_1$ and *Turns*$_{10}$ are the number of guesses needed on the first and on the last task. Standard feedback; mean ± SEM across five seeds. Without a cue no model ever tries a recurring target; with a vague cue Claude Sonnet 4.6 and Gemini 2.5 Flash identify the set, GPT-4o only partially.

C.2 GEMMA3-12B

To test whether the findings depend on the base model, we repeat the multi-environment, latent-shift and environment-shift experiments on Gemma3-12B, using the same models, latent sets, evaluation protocol and aggregation as the corresponding Qwen3-8B tables (the in-distribution comparison is in Table 2). Cross-task RL achieves the highest in-distribution cumulative reward on all six environments. Under multi-environment training, it outperforms single-task RL across all four environments by approximately 28% on average, but shows negligible average improvement over single-environment cross-task RL. Under latent shift, it outperforms the base in cumulative reward in all six environments and single-task RL in five. Under environment shift, transfer is weaker and less consistent than for Qwen3-8B: the leave-one-out model improves cumulative reward on Number Guessing only and matches the base on the other three held-out environments (Table 16). Gain must be read with caution: on Number Guessing the base posts the larger Gain (+46.5% against +33.7%) only because it starts from a lower initial reward (floor effect), while the leave-one-out model has the higher cumulative and final reward; on the other three environments both models start from the same initial reward and their Gains differ by a few points. Judged on cumulative and final-task reward, the metrics we use for OOD-2, cross-task training transfers modestly to held-out environments for Qwen3-8B and less consistently for Gemma3-12B.

C.3 LATENT BREAKDOWN

Tables 2 and 4 report each environment as an average over its latents. Table 17 resolves that average, reporting every latent separately so that per-latent variation is visible. Table 18 gives the same breakdown for Gemma3-12B. Two patterns are worth noting. First, the cross-task advantage is not

		Cross-task (multi env)			Single (multi env)			Cross-task (one env)		
	Env.	Cum.	Gain (%)	Final	Cum.	Gain (%)	Final	Cum.	Gain (%)	Final
Gemma3-12B	Num. guess	**6.7** ±0.02	**+48.8** ±2.6	**0.70** ±0.01	5.4 ±0.04	−37.1 ±1.8	0.42 ±0.01	6.7 ±0.01	+9.4 ±1.4	0.67 ±0.01
	Hangman	9.1 ±0.00	+1.5 ±0.1	0.92 ±0.00	8.3 ±0.03	−7.1 ±0.3	0.81 ±0.01	**9.2** ±0.00	**+6.5** ±0.3	**0.93** ±0.00
	Wordladder	8.7 ±0.03	**+0.3** ±1.2	0.87 ±0.01	7.5 ±0.03	−6.6 ±1.0	0.77 ±0.00	**9.4** ±0.02	−1.6 ±0.7	**0.94** ±0.01
	Secretary	**9.9** ±0.02	+2.9 ±0.2	0.99 ±0.00	6.1 ±0.00	−5.4 ±0.8	0.58 ±0.01	9.6 ±0.00	**+72.6** ±0.1	**1.00** ±0.00

Table 14. Gemma3-12B: multi-environment cross-task RL. Same protocol as Table 3, for Gemma3-12B: the four-environment cross-task and single-task models against the single-environment cross-task models, each environment evaluated on the multi-environment model's two training latents. Each entry is mean ± standard error across 5 independent decoding seeds.

		Cross-task RL			Single-task RL			Base		
	Env.	Cum.	Gain (%)	Final	Cum.	Gain (%)	Final	Cum.	Gain (%)	Final
Gemma3-12B	Num. guess	**5.9** ±0.05	−11.9 ±1.7	0.54 ±0.01	5.3 ±0.06	−47.3 ±4.5	0.35 ±0.03	5.7 ±0.13	**+128.1** ±15.5	**0.58** ±0.04
	Mastermind	**3.9** ±0.03	**−14.1** ±2.4	0.33 ±0.01	3.8 ±0.04	−36.3 ±1.8	0.30 ±0.01	3.5 ±0.04	−14.7 ±3.1	**0.34** ±0.01
	Hangman	**9.0** ±0.01	**+6.5** ±0.5	**0.91** ±0.00	8.1 ±0.03	−11.1 ±1.3	0.81 ±0.01	6.6 ±0.04	−2.6 ±0.9	0.66 ±0.01
	Wordladder	9.4 ±0.02	−6.9 ±0.4	0.90 ±0.01	**9.5** ±0.02	−1.0 ±0.9	**0.95** ±0.01	8.6 ±0.05	**+2.7** ±1.5	0.86 ±0.01
	Secretary	**3.8** ±0.04	+2.7 ±1.1	**0.38** ±0.01	2.8 ±0.00	+7.2 ±0.5	0.27 ±0.00	3.0 ±0.03	**+25.2** ±3.3	0.31 ±0.01
	Wordle	**3.8** ±0.04	−29.9 ±3.7	0.31 ±0.01	3.4 ±0.02	−45.9 ±1.8	**0.32** ±0.01	2.4 ±0.04	**−27.4** ±5.0	0.22 ±0.01

Table 15. Gemma3-12B: generalization under latent shift (OOD-1). Same protocol as Table 4, for Gemma3-12B, evaluated on the single held-out latent per environment.

		Cross-task RL (LOO)			Base		
	Env.	Cum.	Gain (%)	Final	Cum.	Gain (%)	Final
Gemma3-12B	Num. guess	**5.5** ±0.04	+33.7 ±3.6	**0.55** ±0.01	5.3 ±0.08	**+46.5** ±6.4	0.51 ±0.02
	Hangman	7.0 ±0.04	−9.3 ±0.6	0.68 ±0.00	**7.1** ±0.02	**−5.0** ±0.7	**0.70** ±0.00
	Wordladder	**6.2** ±0.05	**−13.2** ±1.7	**0.60** ±0.01	**6.2** ±0.03	−15.7 ±1.7	0.59 ±0.01
	Secretary	4.3 ±0.02	+10.4 ±0.3	**0.43** ±0.00	**4.4** ±0.02	**+10.9** ±0.9	**0.43** ±0.00

Table 16. Gemma3-12B: generalization under environment shift (OOD-2). Same protocol as Table 5, for Gemma3-12B: a leave-one-out cross-task model, trained on all environments *except* the evaluation one, against the base model. Each entry is mean ± standard error across 5 independent decoding seeds.

uniform across latents: on `number_guessing` it is concentrated in `dynamic_range`, where cross-task RL gains +313.3% while both baselines are near zero, and the environment-level Gain in Table 2 largely reflects that one latent. Second, the ordering between arms is stable even where the margins are small: cross-task RL has the highest cumulative reward on 22 of the 24 latents, the two exceptions both being `secretary` (`sorted_order`, where single-task RL saturates the environment at 10.00, and the held-out `best_is_last`).

C.4 Equal Budget Ablations

Both single-task RL and cross-task RL are trained to saturation, i.e. until reward plateaus on held-out validation, so the comparison in Section 4 reflects each recipe's best achievable performance rather than under-training of one. It does not, however, match the two on training budget: cross-task RL is trained on 10-episode sequences and single-task RL on 1-episode rollouts, so cross-task RL sees more task episodes per training step. Two observations suggest the advantage is not baked in during training. As a fairness check, we compare first-episode reward G_1, before any cross-task context has accumulated: the gap is small, while by episode 10 cross-task RL leads clearly and consistently across every environment (Table 19). This is the pattern expected if the two start from comparable baseline capability and diverge as a function of accumulated cross-task experience. The full per-episode trajectory shows the same shape: single-task RL's reward is roughly flat (or slightly declining) across the sequence, consistent with having no mechanism to exploit cross-task context, while cross-task RL's reward rises across episodes (Table 20).

	Latent	Cross-task RL			Single-task RL			Base		
		Cum.	Gain (%)	Final	Cum.	Gain (%)	Final	Cum.	Gain (%)	Final
Num. guess	set_of_3	**6.4** ±0.06	**+66.0** ±7.8	**0.66** ±0.01	4.5 ±0.06	−17.2 ±2.2	0.39 ±0.01	3.2 ±0.10	−26.7 ±4.4	0.23 ±0.02
	set_of_2	**7.8** ±0.07	**+55.9** ±3.5	**0.87** ±0.01	4.2 ±0.03	−20.4 ±1.1	0.38 ±0.00	3.7 ±0.13	−20.3 ±0.9	0.33 ±0.01
	dynamic_range	**3.0** ±0.10	**+313.3** ±41.2	**0.39** ±0.03	0.4 ±0.03	−34.7 ±27.1	0.03 ±0.01	0.3 ±0.03	—	0.03 ±0.01
	range_100†	**6.2** ±0.08	**+47.3** ±14.9	**0.63** ±0.03	3.7 ±0.04	−6.7 ±3.3	0.35 ±0.01	2.8 ±0.04	−35.7 ±6.0	0.22 ±0.02
Mastermind	strictly_ascending	**7.8** ±0.03	**+34.1** ±2.3	**0.85** ±0.01	6.3 ±0.05	−23.9 ±0.9	0.59 ±0.01	4.9 ±0.05	+5.3 ±2.1	0.50 ±0.01
	strictly_descending	**8.4** ±0.03	**+3.9** ±0.6	**0.85** ±0.01	5.2 ±0.05	−22.0 ±0.9	0.50 ±0.01	4.9 ±0.06	+0.8 ±2.6	0.52 ±0.01
	first_is_6	**5.4** ±0.03	−18.8 ±1.2	**0.49** ±0.01	4.0 ±0.04	**−9.0** ±1.2	0.41 ±0.01	3.3 ±0.06	−12.1 ±2.8	0.34 ±0.01
	has_pair†	**4.3** ±0.01	**+16.6** ±2.0	**0.44** ±0.01	3.5 ±0.03	−10.0 ±1.7	0.35 ±0.01	3.4 ±0.02	−6.0 ±3.3	0.36 ±0.01
Hangman	vowel_count_4	**7.8** ±0.07	**+3.1** ±0.8	**0.81** ±0.01	7.0 ±0.04	−6.8 ±1.9	0.71 ±0.01	4.0 ±0.11	−5.2 ±3.8	0.40 ±0.02
	has_double_letter	**5.9** ±0.07	−10.8 ±0.7	**0.52** ±0.01	5.0 ±0.07	−8.4 ±3.2	0.49 ±0.01	3.5 ±0.06	**−7.2** ±5.7	0.33 ±0.01
	starts_with_S	**5.7** ±0.08	+1.6 ±4.8	**0.53** ±0.02	4.9 ±0.05	+8.5 ±3.4	0.50 ±0.01	3.1 ±0.05	**+24.7** ±5.3	0.32 ±0.01
	ending_ABLE†	**6.7** ±0.02	+26.8 ±1.1	**0.70** ±0.00	5.9 ±0.03	+16.5 ±1.3	0.60 ±0.01	5.6 ±0.03	**+69.1** ±8.3	0.59 ±0.00
Wordladder	hub_word_4letter	**8.3** ±0.05	**+3.2** ±1.8	**0.85** ±0.01	7.3 ±0.06	+1.0 ±2.7	0.77 ±0.01	3.4 ±0.06	−4.0 ±4.1	0.35 ±0.01
	hub_word_3letter	**9.9** ±0.02	**−0.6** ±0.5	**0.98** ±0.00	9.2 ±0.04	−8.3 ±1.9	0.88 ±0.01	4.9 ±0.11	−16.2 ±4.1	0.48 ±0.03
	order_outside_in	**10.0** ±0.00	**−0.1** ±0.0	**1.00** ±0.00	9.6 ±0.02	−3.2 ±1.0	0.95 ±0.01	6.7 ±0.04	−0.5 ±3.3	0.66 ±0.02
	order_left_to_right†	**9.6** ±0.02	−2.4 ±0.5	**0.94** ±0.01	9.3 ±0.02	**−0.0** ±1.4	0.92 ±0.01	7.1 ±0.03	−1.9 ±1.1	0.73 ±0.01
Secretary	inverse_order	**4.6** ±0.05	**+175.5** ±3.3	**0.49** ±0.01	3.6 ±0.01	+7.6 ±0.2	0.36 ±0.00	1.6 ±0.01	+0.3 ±0.1	0.16 ±0.00
	fixed_position_2	**9.8** ±0.00	+10.7 ±0.1	**1.00** ±0.00	7.7 ±0.01	+1.4 ±1.0	0.81 ±0.01	8.0 ±0.01	**+29.6** ±1.0	0.79 ±0.00
	sorted_order	9.0 ±0.09	**+4.1** ±0.5	0.91 ±0.01	**10.0** ±0.00	+0.0 ±0.0	**1.00** ±0.00	4.3 ±0.03	+1.5 ±0.7	0.43 ±0.00
	best_is_last†	4.2 ±0.04	+5.3 ±1.3	0.39 ±0.00	**4.4** ±0.01	+6.0 ±1.3	0.42 ±0.00	4.4 ±0.02	**+40.3** ±2.8	**0.45** ±0.01
Wordle	vowel_count_3	**6.8** ±0.01	**+14.8** ±1.7	**0.67** ±0.01	3.1 ±0.03	−49.6 ±0.4	0.27 ±0.00	2.9 ±0.05	−0.2 ±2.9	0.28 ±0.01
	vowels_clustered	**6.4** ±0.03	**+19.1** ±2.9	**0.65** ±0.02	2.8 ±0.03	−64.6 ±1.7	0.22 ±0.01	2.1 ±0.03	+11.6 ±4.6	0.23 ±0.01
	position_E_at_1	**5.3** ±0.01	+4.6 ±1.3	**0.53** ±0.01	3.4 ±0.03	−55.4 ±1.0	0.28 ±0.01	2.5 ±0.04	**+16.1** ±4.4	0.26 ±0.00
	has_double_letter†	**4.9** ±0.03	**+1.8** ±3.1	**0.50** ±0.01	2.9 ±0.03	−49.5 ±0.5	0.26 ±0.00	2.0 ±0.03	−43.5 ±2.6	0.17 ±0.01

Table 17. Per-latent breakdown (Qwen3-8B). Tables 2 and 4 resolved to individual latents. Rows marked † are the held-out (OOD-1) latent of each environment, which the single-env models never trained on; the other three per environment are the in-distribution training latents. Each entry is one latent with 50 trajectories, reported as mean ± standard error across 5 independent decoding seeds. Bold marks the best arm in each row and metric. A dash denotes an undefined Gain, where the first-episode reward is 0 and $(r_N - r_1)/r_1$ does not exist.

	Latent	Cross-task RL			Single-task RL			Base		
		Cum.	Gain (%)	Final	Cum.	Gain (%)	Final	Cum.	Gain (%)	Final
Num. guess	set_of_3	**7.5** ±0.04	**+30.4** ±2.9	**0.80** ±0.01	4.7 ±0.04	−61.7 ±2.1	0.29 ±0.02	5.0 ±0.07	+0.1 ±4.2	0.44 ±0.01
	set_of_2	**8.0** ±0.03	**+30.0** ±1.6	**0.85** ±0.01	5.5 ±0.13	−37.1 ±4.6	0.45 ±0.03	5.7 ±0.05	+23.7 ±2.3	0.56 ±0.02
	dynamic_range	**2.5** ±0.11	+395.6 ±117.3	**0.26** ±0.02	1.0 ±0.05	+38.3 ±63.0	0.05 ±0.02	1.3 ±0.04	**+412.2** ±182.6	0.12 ±0.02
	range_100†	**5.9** ±0.05	−11.9 ±1.7	0.54 ±0.01	5.3 ±0.06	−47.3 ±4.5	0.35 ±0.03	5.7 ±0.13	**+128.1** ±15.5	**0.58** ±0.04
Mastermind	strictly_ascending	**9.4** ±0.03	**−7.5** ±1.2	**0.90** ±0.01	5.3 ±0.03	−30.4 ±1.9	0.48 ±0.01	5.6 ±0.05	−16.2 ±1.3	0.55 ±0.01
	strictly_descending	**8.6** ±0.07	**−7.4** ±0.9	**0.79** ±0.00	6.0 ±0.08	−26.8 ±4.1	0.53 ±0.02	4.8 ±0.03	−12.0 ±3.9	0.47 ±0.02
	first_is_6	**5.7** ±0.03	−8.6 ±1.8	**0.51** ±0.01	5.2 ±0.08	−27.9 ±1.6	0.47 ±0.01	3.8 ±0.04	**−2.6** ±2.8	0.39 ±0.02
	has_pair†	**3.9** ±0.03	**−14.1** ±2.4	0.33 ±0.01	3.8 ±0.04	−36.3 ±1.8	0.30 ±0.01	3.5 ±0.04	−14.7 ±3.1	**0.34** ±0.01
Hangman	vowel_count_4	**9.4** ±0.00	**+6.5** ±0.2	**0.95** ±0.00	8.9 ±0.01	−7.1 ±0.3	0.88 ±0.00	7.6 ±0.01	−7.0 ±0.7	0.75 ±0.01
	has_double_letter	**8.4** ±0.01	**−1.0** ±1.0	**0.81** ±0.01	8.2 ±0.03	−11.9 ±2.0	0.78 ±0.02	6.3 ±0.06	−17.7 ±1.3	0.59 ±0.01
	starts_with_S	**9.0** ±0.01	**+5.1** ±0.6	**0.91** ±0.00	8.7 ±0.01	−5.7 ±0.4	0.86 ±0.00	6.9 ±0.07	−12.2 ±0.6	0.65 ±0.01
	ending_ABLE†	**9.0** ±0.01	**+6.5** ±0.5	**0.91** ±0.00	8.1 ±0.03	−11.1 ±1.3	0.81 ±0.01	6.6 ±0.04	−2.6 ±0.9	0.66 ±0.01
Wordladder	hub_word_4letter	**9.0** ±0.02	**−1.9** ±1.4	**0.89** ±0.01	7.9 ±0.04	−9.2 ±1.1	0.78 ±0.00	5.5 ±0.06	−10.6 ±2.4	0.55 ±0.01
	hub_word_3letter	**9.8** ±0.02	**−1.2** ±0.4	**0.98** ±0.00	9.5 ±0.03	−6.4 ±1.3	0.93 ±0.01	7.0 ±0.03	−19.5 ±2.7	0.64 ±0.01
	order_outside_in	**10.0** ±0.00	**+0.0** ±0.0	**1.00** ±0.00	9.9 ±0.01	−1.0 ±0.4	0.99 ±0.00	8.4 ±0.01	−3.8 ±1.0	0.80 ±0.01
	order_left_to_right†	9.4 ±0.02	−6.9 ±0.4	0.90 ±0.00	**9.5** ±0.02	−1.0 ±0.9	**0.95** ±0.01	8.6 ±0.05	**+2.7** ±1.5	0.86 ±0.01
Secretary	inverse_order	9.2 ±0.00	**+459.5** ±1.5	**1.00** ±0.00	**9.6** ±0.02	−4.4 ±0.0	0.96 ±0.00	1.4 ±0.00	+0.0 ±0.0	0.14 ±0.00
	fixed_position_2	**10.0** ±0.00	+2.0 ±0.0	**1.00** ±0.00	2.8 ±0.01	+5.4 ±0.4	0.26 ±0.00	7.3 ±0.04	**+13.3** ±1.1	0.72 ±0.01
	sorted_order	**10.0** ±0.00	+0.0 ±0.0	**1.00** ±0.00	0.3 ±0.00	**+26.4** ±0.0	0.03 ±0.00	4.5 ±0.00	+0.2 ±0.0	0.45 ±0.00
	best_is_last†	**3.8** ±0.04	+2.7 ±1.1	**0.38** ±0.01	2.8 ±0.00	+7.2 ±0.5	0.27 ±0.00	3.0 ±0.03	**+25.2** ±3.3	0.31 ±0.01
Wordle	vowel_count_3	**9.0** ±0.06	**−1.4** ±1.1	**0.89** ±0.01	3.8 ±0.05	−61.7 ±0.8	0.31 ±0.01	2.4 ±0.04	−22.0 ±4.7	0.24 ±0.01
	vowels_clustered	**5.1** ±0.03	−32.7 ±1.2	**0.43** ±0.01	4.1 ±0.03	−53.3 ±2.5	0.36 ±0.02	2.3 ±0.04	**−24.0** ±4.8	0.21 ±0.01
	position_E_at_1	**5.9** ±0.04	**−12.7** ±1.2	**0.61** ±0.01	3.1 ±0.02	−50.0 ±2.6	0.31 ±0.02	2.2 ±0.03	−16.7 ±5.6	0.21 ±0.01
	has_double_letter†	**3.8** ±0.04	−29.9 ±3.7	0.31 ±0.01	3.4 ±0.02	−45.9 ±1.8	**0.32** ±0.01	2.4 ±0.04	**−27.4** ±5.0	0.22 ±0.01

Table 18. Per-latent breakdown (Gemma3-12B). Table 2 (Gemma3-12B block) and Table 15 resolved to individual latents. Rows marked † are the held-out (OOD-1) latent of each environment, which the single-env models never trained on; the other three per environment are the in-distribution training latents. Each entry is one latent with 50 trajectories, reported as mean ± standard error across 5 independent decoding seeds. Bold marks the best arm in each row and metric. A dash denotes an undefined Gain, where the first-episode reward is 0 and $(r_N - r_1)/r_1$ does not exist.

An equal-budget ablation is the direct way to stress-test the comparison. We add a control that matches single-task RL to cross-task RL on the total task episodes and instances seen during training. From a single pool of 10-episode sequences, we build a matched single-task dataset that uses the same underlying task instances but trains on them as 1-episode rollouts, so single-task RL sees every task cross-task RL sees. To equalize total episodes, single-task RL is trained for 5× more gradient steps with 2× the batch size (more stable for single-task training), so it sees $2 \times 5 = 10\times$ more

episodes, exactly matching cross-task RL's 10-episode sequences. Both recipes therefore see the same total number of task episodes, drawn from the same underlying task instances.

Table 21 reports results with Qwen3-8B on three settings, number guessing on a single latent, Wordle on its three training latents, and Wordle on the held-out latent, and on an additional setting, a model trained on three environments (Word Ladder, Secretary and Number Guessing) with two latents each, both in distribution and on the held-out latents. The same pattern holds throughout: cross-task RL retains a clear advantage over budget-matched single-task RL.

We also carried out the agent-switching experiment with the budget-matched single-task RL baseline, on number guessing with a single latent (Table 22), with $\mathrm{ExploreGain}$ and $\mathrm{ExploitGain}$ as defined in Section 2 and budget-matched single-task RL in place of ST. Even against the budget-matched baseline, cross-task RL exploits the same replayed context better in nearly all settings ($\mathrm{ExploitGain}$ of $+4\%$ to $+16\%$ for $K = 2, 4, 6$, under both CT- and ST-generated contexts). The gap narrows at $K = 8$: with only two episodes remaining there is little room for differentiation, and in this simple single-latent setting the latent is typically already fully revealed in context by then. The positive $\mathrm{ExploreGain}$ values further indicate that cross-task RL also gathers more informative context during the first K episodes.

The number of tokens processed during training cannot be set in advance: it depends on the agent's own policy, that is, how many turns it takes to solve each episode and how many tokens it uses per turn. What we can hold fixed is the number of task episodes and task instances seen during training, and we match the two recipes on this budget. That said, the measured total training tokens are comparable, with modest differences of no consistent sign: on number guessing and Hangman, equal-budget single-task RL processes somewhat more tokens than cross-task RL (246M vs. 192M, and 224M vs. 191M, respectively), while on Wordle it processes somewhat fewer (441M vs. 516M). Extending the equal-budget control to the remaining environments is left to future work.

Env.	ΔG_1 (cross − single)	ΔG_{10} (cross − single)
Num. guess	$+0.020 \pm 0.015$	$+0.373 \pm 0.012$
Mastermind	$+0.060 \pm 0.007$	$+0.227 \pm 0.010$
Hangman	$+0.044 \pm 0.006$	$+0.053 \pm 0.012$
Wordladder	$+0.035 \pm 0.006$	$+0.076 \pm 0.006$
Secretary	-0.061 ± 0.004	$+0.076 \pm 0.005$
Wordle	-0.056 ± 0.005	$+0.359 \pm 0.005$

Table 19. First- vs last-episode gap (Qwen3-8B). Difference in mean episode reward between cross-task RL and single-task RL on the first episode (G_1, before any cross-task context has accumulated) and on the last episode (G_{10}), on the three in-distribution latents of each environment with 50 trajectories per latent; mean ± standard error across 5 independent decoding seeds.

Model	Episode									
	1	2	3	4	5	6	7	8	9	10
Single-task RL	0.331	0.291	0.317	0.352	0.304	0.292	0.314	0.288	0.302	0.266
Cross-task RL	0.352	0.513	0.551	0.606	0.621	0.632	0.623	0.606	0.586	0.639

Table 20. Per-episode reward on number guessing (Qwen3-8B). Mean reward on each of the 10 episodes of the sequence, averaged over the three in-distribution latents, 50 trajectories per latent and 5 decoding seeds.

C.5 Multiple Training Seeds

Unless otherwise specified, reported uncertainties are standard errors across five decoding seeds and quantify evaluation variability for a fixed trained model. To assess training stability, we additionally train each recipe with three independent seeds on Qwen3-8B Number Guessing. Given the cost of repeated RL training and the approximately 11,000 A100-80GB GPU-hours used across our experiments, we restrict this additional study to one setting. Cross-task RL outperforms single-task RL on cumulative reward, Gain, and final-task reward across all three seeds (Table 23).

Setting	Model	Cum.	Gain (%)	Final
Num. guess, `set_of_3`	Cross-task RL	**6.9** ±0.05	**+9.2** ±1.7	**0.71** ±0.01
	Single-task RL (equal budget)	6.1 ±0.12	−4.2 ±3.6	0.63 ±0.02
Wordle, 3 ID latents	Cross-task RL	**6.1** ±0.01	**+13.1** ±1.5	**0.62** ±0.01
	Single-task RL (equal budget)	3.4 ±0.02	−61.2 ±0.9	0.28 ±0.01
Wordle, held-out (OOD-1)	Cross-task RL	**4.9** ±0.03	**+1.8** ±3.1	**0.50** ±0.01
	Single-task RL (equal budget)	2.9 ±0.04	−59.3 ±1.7	0.23 ±0.01
3Envs, trained latents	Cross-task RL	**9.0** ±0.01	**+13.5** ±0.3	**0.91** ±0.00
	Single-task RL (equal budget)	7.7 ±0.01	−11.2 ±0.6	0.72 ±0.00
3Envs, held-out (OOD-1)	Cross-task RL	**4.9** ±0.03	**+16.0** ±1.3	**0.50** ±0.01
	Single-task RL (equal budget)	4.3 ±0.03	−4.8 ±1.9	0.40 ±0.01

Table 21. Equal-budget ablation (Qwen3-8B). Cross-task RL against a budget-matched single-task RL model trained on the same task instances as 1-episode rollouts with 5× the gradient steps and 2× the batch size, so that both see the same total number of task episodes. *3Envs* is a model trained jointly on Word Ladder, Secretary and Number Guessing with two latents each, evaluated on its trained latents and on the held-out latents of those environments. Each entry averages over the listed latents and 50 trajectories per latent, reported as mean ± standard error across 5 independent decoding seeds.

Switch point K	Reference C	ExploreGain_C	ExploitGain_C
2	$C = \text{CT}$	+1.3% ±0.7	+15.1% ±1.1
	$C = \text{ST}$	+2.0% ±2.8	+15.9% ±3.2
4	$C = \text{CT}$	+8.0% ±1.5	+5.5% ±1.7
	$C = \text{ST}$	+9.8% ±4.3	+7.2% ±3.1
6	$C = \text{CT}$	+8.2% ±2.2	+4.0% ±0.2
	$C = \text{ST}$	+11.2% ±3.0	+6.9% ±1.3
8	$C = \text{CT}$	+17.6% ±2.5	+1.6% ±1.2
	$C = \text{ST}$	+13.0% ±2.4	−2.4% ±1.6

Table 22. Agent switching against the budget-matched single-task RL baseline (Qwen3-8B). Same protocol as Table 5, on `number_guessing` with the single latent `set_of_3`, with ST the equal-budget single-task RL model of Table 21. Each entry pools 50 trajectories and is reported as mean ± standard error across 5 independent decoding seeds.

Model	Training seed	Cum.	Gain (%)	Final
Cross-task RL	263	5.7 ±0.04	+82.6 ±5.7	0.64 ±0.01
	390	6.5 ±0.06	+43.9 ±2.5	0.73 ±0.01
	60	5.7 ±0.02	+38.0 ±2.5	0.62 ±0.01
	mean (n=3)	6.0 ±0.27	+54.8 ±14.0	0.66 ±0.03
Single-task RL	263	3.1 ±0.03	−19.7 ±1.2	0.27 ±0.00
	390	3.8 ±0.05	−1.2 ±1.4	0.39 ±0.00
	60	3.5 ±0.05	−11.8 ±1.4	0.31 ±0.00
	mean (n=3)	3.5 ±0.23	−10.9 ±5.3	0.32 ±0.04

Table 23. Training-seed stability on number guessing (Qwen3-8B). Cross-task RL and single-task RL, both Qwen3-8B, each trained with three independent training seeds; seed 263 is the released model. Each row averages over the three in-distribution training latents and 50 trajectories per latent, reported as mean ± standard error across 5 independent decoding seeds; the mean rows report mean ± standard error across the three training seeds.

C.6 Shared Latent Ablation

In Table 2, the tasks within a sequence share a latent. To test whether this shared structure is what cross-task RL exploits, we evaluate the cross-task RL model of each family on the same in-

distribution tasks, re-dealt across sequences so that no sequence shares a latent. We evaluate the re-dealt sequences both with the original system prompt and with the cross-task hint removed from it, since the tasks then share no property; everything else follows the protocol of Table 2. For both model families and under either prompt, removing the shared latent lowers cumulative and final reward in every environment except Wordladder and, for Gemma3-12B, Hangman, where the model is already close to the maximum (Table 24). Qwen3-8B on Wordle is the other exception: it is essentially unaffected by removing the latent, and we have reason to believe this is rooted in the intrinsic additional difficulty of Wordle with respect to the other games rather than in the shared structure.

		Shared latent			No shared latent, same prompt			No shared latent, no cross-task hint		
	Env.	Cum.	Gain (%)	Final	Cum.	Gain (%)	Final	Cum.	Gain (%)	Final
Qwen3-8B	Num. guess	**5.7** ±0.04	**+82.6** ±5.7	**0.64** ±0.01	3.4 ±0.03	−4.7 ±3.1	0.34 ±0.01	4.1 ±0.02	+8.0 ±1.1	0.42 ±0.00
	Mastermind	**7.2** ±0.02	**+6.6** ±0.9	**0.73** ±0.01	6.5 ±0.01	−13.3 ±0.8	0.58 ±0.00	6.5 ±0.01	−9.6 ±0.7	0.61 ±0.01
	Hangman	**6.4** ±0.04	**−1.7** ±1.7	**0.62** ±0.01	6.2 ±0.03	+3.2 ±1.7	0.60 ±0.01	5.5 ±0.03	−15.3 ±2.1	0.49 ±0.01
	Wordladder	**9.4** ±0.02	**+0.7** ±0.6	**0.94** ±0.00	9.3 ±0.01	+0.5 ±0.5	0.93 ±0.00	9.3 ±0.01	−0.2 ±1.1	0.93 ±0.00
	Secretary	**7.8** ±0.04	**+22.8** ±0.4	**0.80** ±0.00	6.9 ±0.02	+7.8 ±1.0	0.69 ±0.01	6.4 ±0.01	+15.3 ±0.3	0.62 ±0.00
	Wordle	**6.1** ±0.01	**+13.1** ±1.5	**0.62** ±0.01	**6.1** ±0.01	**+11.0** ±1.5	**0.62** ±0.00	**6.1** ±0.02	**+14.2** ±1.3	**0.63** ±0.01
Gemma3-12B	Num. guess	**6.0** ±0.04	**+43.2** ±1.5	**0.64** ±0.01	4.4 ±0.02	−10.5 ±2.9	0.42 ±0.01	4.4 ±0.03	−7.2 ±2.2	0.43 ±0.01
	Mastermind	**8.1** ±0.03	**−3.5** ±0.6	**0.76** ±0.01	6.8 ±0.04	−31.7 ±0.4	0.53 ±0.00	6.9 ±0.02	−27.6 ±0.5	0.55 ±0.00
	Hangman	8.9 ±0.00	+3.6 ±0.5	0.89 ±0.00	**9.0** ±0.01	**+7.1** ±0.3	**0.90** ±0.00	8.9 ±0.02	+8.3 ±0.2	0.90 ±0.00
	Wordladder	**9.6** ±0.01	**−1.0** ±0.5	**0.96** ±0.00	**9.6** ±0.02	**−1.3** ±0.4	**0.96** ±0.00	**9.6** ±0.01	**−1.3** ±0.5	**0.96** ±0.00
	Secretary	**9.7** ±0.00	**+39.0** ±0.0	**1.00** ±0.00	6.7 ±0.01	−7.4 ±0.3	0.65 ±0.00	6.8 ±0.02	+2.8 ±0.5	0.70 ±0.00
	Wordle	**6.7** ±0.01	**−13.9** ±0.5	**0.64** ±0.00	5.7 ±0.03	−33.6 ±1.0	0.50 ±0.01	5.7 ±0.06	−33.4 ±0.9	0.49 ±0.01

Table 24. Cross-task RL without a shared latent. The cross-task RL model of each family evaluated on the in-distribution latents (left, as in Table 2) and on the same tasks re-dealt so that no sequence shares a latent, either with the same system prompt (middle) or with the cross-task hint removed from the prompt (right). *Cum.*, *Gain* and *Final* are as in Table 2; mean ± standard error across 5 decoding seeds, 150 sequences per cell.

C.7 Experience Collection vs. Utilization: Additional Results

In this appendix we show additional results for the experience collection vs. utilization analysis in Section 4 for Number Guessing. For all experiments we follow the same agent-switching protocol. First, we compare Cross-Task RL against the *base* Qwen3-8B model on Number Guessing, alongside the corresponding agent-switch trajectory plot. Second, we report the Cross-Task RL vs. Single-task RL agent-switching tables for three additional environments (Hangman, Secretary, and Word Ladder) to test whether the collection/utilization decomposition behaves consistently across the benchmark.

Number Guessing: Cross-Task RL vs. base. Table 25 reports the same protocol as Section 4 but with the base Qwen3-8B model in place of Single-task RL. Both gaps are dramatically larger than against Single-task RL, reaching $+172.4\%$ in ExploreGain and $+196.7\%$ in ExploitGain. Figure 8 shows the corresponding trajectories.

Hangman. Table 26 shows that on Hangman, Cross-Task RL leads on both gains, with ExploreGain of $+24\%$ to $+63\%$ and ExploitGain of $+7\%$ to $+38\%$. Unlike Number Guessing, from $K = 4$ on the collection gain is the larger of the two, and it grows with K while the utilization gain shrinks.

Secretary. Table 27 shows a more balanced picture: ExploreGain of $+7\%$ to $+37\%$ and ExploitGain of $+11\%$ to $+30\%$, with neither gain consistently larger across switch points.

Word Ladder. Table 28 shows that on Word Ladder both gains are small (ExploreGain within $\pm 1\%$, ExploitGain at most $+5\%$), reflecting the convergence of Cross-Task RL and Single-task RL on this environment noted in Section 4.

Two patterns hold across environments. First, Cross-Task RL posts non-negative gains on both collection and utilization in almost every cell; the exceptions are the small negative ExploreGain values on Word Ladder. Second, which gain is larger depends on the environment: utilization on

Number Guessing, collection on Hangman, mixed on Secretary, neither on Word Ladder. As in Section 4, both gains are relative to the reference agent and switch point rather than pure measures of either ability.

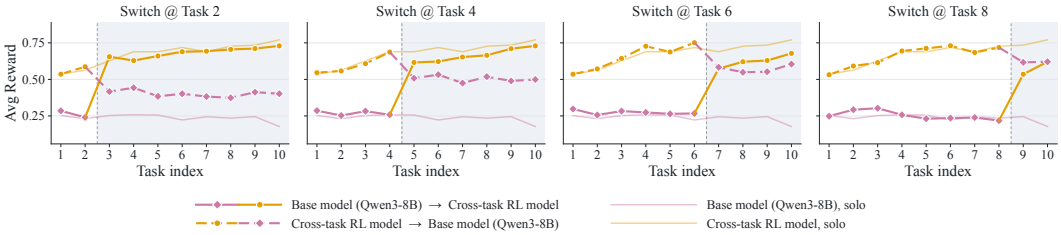


Figure 8. Agent-switching: Cross-task RL vs. base Qwen3-8B on Number Guessing. The same asymmetry holds: Cross-task RL recovers most of its solo trajectory even when the early context was built by the base model, while the base model cannot exploit a context built by Cross-task RL.

Switch point K	Reference C	ExploreGain_C	ExploitGain_C
2	$C = \text{CT}$	$+3.7\%$ ± 0.9	$+74.2\%$ ± 2.0
	$C = \text{Base}$	$+69.2\%$ ± 4.1	$+184.4\%$ ± 7.3
4	$C = \text{CT}$	$+6.2\%$ ± 0.5	$+41.7\%$ ± 0.8
	$C = \text{Base}$	$+122.4\%$ ± 4.7	$+196.7\%$ ± 7.4
6	$C = \text{CT}$	$+14.2\%$ ± 0.7	$+27.4\%$ ± 0.7
	$C = \text{Base}$	$+156.9\%$ ± 5.8	$+186.7\%$ ± 6.5
8	$C = \text{CT}$	$+26.4\%$ ± 1.1	$+21.0\%$ ± 1.4
	$C = \text{Base}$	$+172.4\%$ ± 8.0	$+160.9\%$ ± 8.4

Table 25. Cross-Task RL vs. Base model on `number_guessing` (Qwen3-8B) (4-latent average, with standard errors). Cross-Task RL leads on both gains by a wide margin: large positive values throughout, with the largest gaps appearing when the Base is held fixed in the other role. Each entry pools all four `number_guessing` latents with 50 trajectories each, and is reported as mean $\pm$ standard error across 5 independent decoding seeds.

Switch point K	Reference C	ExploreGain_C	ExploitGain_C
2	$C = \text{CT}$	$+24.2\%$ ± 0.6	$+27.0\%$ ± 1.0
	$C = \text{ST}$	$+34.7\%$ ± 0.5	$+37.7\%$ ± 1.2
4	$C = \text{CT}$	$+35.7\%$ ± 0.6	$+10.1\%$ ± 0.7
	$C = \text{ST}$	$+55.3\%$ ± 1.6	$+25.9\%$ ± 0.7
6	$C = \text{CT}$	$+45.6\%$ ± 1.0	$+8.1\%$ ± 0.7
	$C = \text{ST}$	$+58.5\%$ ± 1.8	$+17.6\%$ ± 0.8
8	$C = \text{CT}$	$+46.9\%$ ± 0.8	$+6.8\%$ ± 1.0
	$C = \text{ST}$	$+63.0\%$ ± 2.8	$+18.4\%$ ± 1.1

Table 26. Cross-Task RL vs. Single-task RL on `hangman` (Qwen3-8B) (4-latent average). All values positive: Cross-Task RL leads on both collection and utilization, with ExploreGain growing in K and ExploitGain shrinking as the post-switch tail shrinks. Each entry pools all four `hangman` latents with 50 trajectories each, and is reported as mean $\pm$ standard error across 5 independent decoding seeds.

Switch point K	Reference C	ExploreGain_C	ExploitGain_C
2	$C = \text{CT}$	$+7.2\%$ ± 0.1	$+28.6\%$ ± 0.5
	$C = \text{ST}$	$+8.4\%$ ± 0.5	$+29.9\%$ ± 0.2
4	$C = \text{CT}$	$+15.4\%$ ± 0.2	$+22.6\%$ ± 0.4
	$C = \text{ST}$	$+15.6\%$ ± 0.3	$+22.8\%$ ± 0.2
6	$C = \text{CT}$	$+19.5\%$ ± 0.3	$+10.9\%$ ± 0.2
	$C = \text{ST}$	$+32.7\%$ ± 0.3	$+23.2\%$ ± 0.4
8	$C = \text{CT}$	$+24.0\%$ ± 0.2	$+17.5\%$ ± 0.3
	$C = \text{ST}$	$+37.2\%$ ± 0.5	$+30.0\%$ ± 0.8

Table 27. Cross-Task RL vs. Single-task RL on `secretary` (Qwen3-8B) (4-latent average). All values positive: Cross-Task RL leads on both collection and utilization, with ExploreGain growing steadily in K. Each entry pools all four `secretary` latents with 50 trajectories each, and is reported as mean $\pm$ standard error across 5 independent decoding seeds.

Switch point K	Reference C	ExploreGain_C	ExploitGain_C
2	$C = \text{CT}$	-0.3% ± 0.2	$+3.7\%$ ± 0.3
	$C = \text{ST}$	$+0.3\%$ ± 0.2	$+4.3\%$ ± 0.1
4	$C = \text{CT}$	-0.2% ± 0.2	$+4.1\%$ ± 0.4
	$C = \text{ST}$	-0.1% ± 0.4	$+4.2\%$ ± 0.2
6	$C = \text{CT}$	-0.3% ± 0.2	$+2.8\%$ ± 0.3
	$C = \text{ST}$	$+1.1\%$ ± 0.3	$+4.2\%$ ± 0.1
8	$C = \text{CT}$	-0.9% ± 0.3	$+3.9\%$ ± 0.7
	$C = \text{ST}$	$+0.5\%$ ± 0.6	$+5.3\%$ ± 0.4

Table 28. Cross-Task RL vs. Single-task RL on `wordladder` (Qwen3-8B) (4-latent average). Differences are uniformly small on this environment ($\leq 5\%$ throughout) and ExploreGain is near zero or slightly negative: both RL variants converge to similar trajectories, leaving little room for cross-task adaptation to matter. Each entry pools all four `wordladder` latents with 50 trajectories each, and is reported as mean $\pm$ standard error across 5 independent decoding seeds.

C.8 Feedback impact on training and generalization

We train Qwen3-8B on four (prompt, feedback) combinations: {full info,some info}$\times${information, standard} (*information* feedback is the condition called *full feedback* in Section 4), with three random seeds each (12 runs total) and evaluate every checkpoint on the full 4×8 grid of (eval prompt, eval feedback)$\times$(8 env–latents). All curves below are mean $\pm$ 1 standard error of the mean across seed samples; the eval cells contributing to each curve are averaged together as described in the captions. Tables 29 and 30 report the corresponding final-step values.

Effect of the training prompt. Figures 9 and 10 present the same data with the panel/line roles swapped. In Figure 9 each panel fixes a training prompt and overlays the two eval prompts; full-info-trained models display a clear in-distribution advantage (the solid "Eval prompt: Full info" curve sits well above the dashed "Eval prompt: Some info" curve), whereas some-info-trained models track essentially the same trajectory under both eval prompts. In Figure 10 each panel fixes an eval prompt and overlays the two training prompts; full-info-trained models lead on full-info eval, while on some-info eval the some-info-trained model is slightly ahead. The takeaway is that training under the more informative prompt buys a sizeable in-distribution gain but a modest cross-prompt deficit, while training under the less informative prompt produces a more robust agent that pays only a small price under the matched-prompt condition.

Effect of the training feedback. Figures 11 and 12 repeat the analysis along the feedback dimension. The pattern is asymmetric: under both eval feedbacks, the standard-feedback-trained model is above the information-feedback-trained model, and the gap between matched- and mismatched-eval-feedback within an information-trained model ($8.940 \rightarrow 8.431$) is much larger than within

Train prompt	Eval prompt	
	Full info	Some info
Full info	8.979 ± 0.107	8.616 ± 0.084
Some info	8.855 ± 0.140	8.860 ± 0.143

Table 29. Average score (cumulative reward summed over 10 episodes) at final training step (Qwen3-8B), broken down by training prompt vs. evaluation prompt. Each cell is the mean $\pm$ 1 standard error across $n = 6$ independent training runs (2 training feedbacks $\times$ 3 seeds), averaged over both eval feedbacks and 8 env–latents.

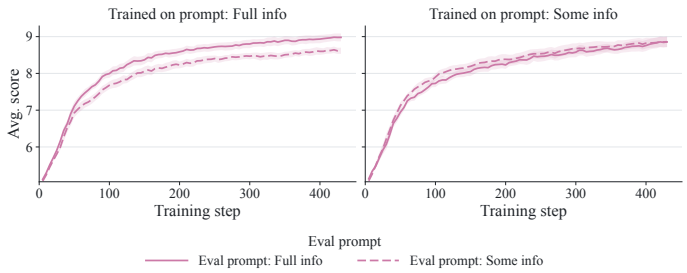


Figure 9. For each training prompt (panels), eval-time score under the two eval prompts (lines), pooled over both the feedbacks (Qwen3-8B). Lines are mean $\pm$ 1 SEM across $n = 6$ training runs (2 feedbacks $\times$ 3 seeds).

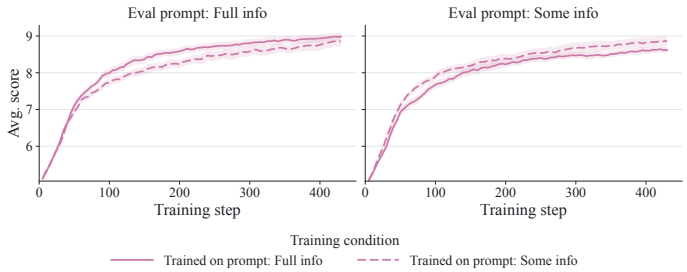


Figure 10. Roles reversed from Figure 9: each panel fixes an eval prompt and compares the two training prompts (Qwen3-8B). Same aggregation and band semantics.

a standard-trained model ($9.044 \rightarrow 8.895$). Training with extra information feedback therefore looks helpful when the same feedback is available at eval time but degrades transfer to the standard-feedback regime, whereas training under the standard feedback transfers well.

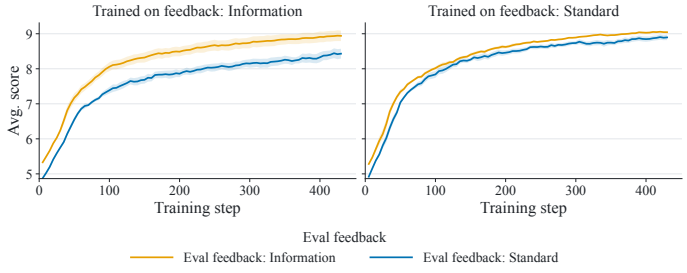


Figure 11. For each training feedback (panels), eval-time score under the two eval feedbacks (lines), pooled over both training prompts, both eval prompts, and 8 env–latents (Qwen3-8B). Lines are mean $\pm$ 1 standard error across $n = 6$ training runs.

Joint (prompt, feedback) view. Figures 13 and 14 avoid pooling over either training axis and present the full 4×4 table of training$\times$eval (prompt, feedback) combinations. Figure 13 shows the four-way trajectory of a single trained model across all eval conditions; on the other hand Figure 14 shows, for a fixed eval condition, which training condition reaches the highest score. The

Train feedback	Eval feedback	
	Information	Standard
Information	8.940 ± 0.149	8.431 ± 0.140
Standard	9.044 ± 0.020	8.895 ± 0.078

Table 30. Average score at final training step (Qwen3-8B), broken down by training feedback vs. evaluation feedback. Each cell is the mean $\pm$ 1 standard error across $n = 6$ independent training runs (2 training prompts $\times$ 3 seeds), averaged over both eval prompts and 8 env–latents.

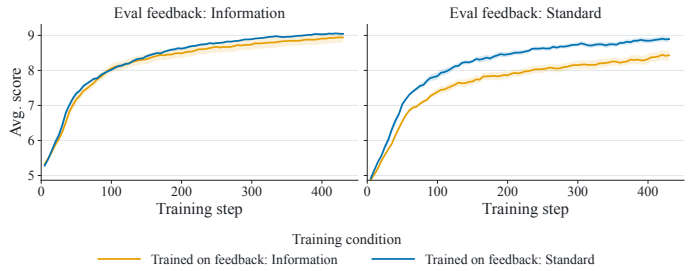


Figure 12. Roles reversed: each panel fixes an eval feedback and compares the two training feedbacks (Qwen3-8B). Same aggregation and band semantics.

dominant factor is the eval feedback itself: information-feedback evaluation produces higher scores than standard-feedback evaluation for every training combination, with gaps ranging from $+0.04$ to $+0.52$. Holding the eval condition fixed, no single training condition wins everywhere, but (some info, standard) is the top-ranked training condition in three of the four eval panels (Figure 14); the exception is the (full info, information) eval panel, where *(full info, standard)* ranks first. Prompt alignment has a secondary effect (matched-prompt cells average 0.18 points above mismatched-prompt cells).

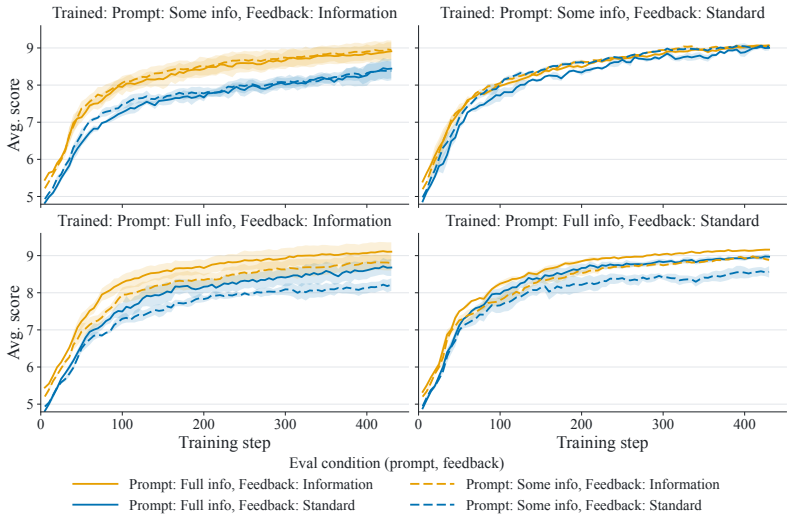


Figure 13. For each training (prompt, feedback) combination (panels), eval-time score under all four eval (prompt, feedback) combinations (lines), averaged over the 8 env–latents (Qwen3-8B). Lines are mean $\pm$ 1 standard error across $n = 3$ seeds.

D Anecdotes for Failures of SOTA models

In this appendix we present a representative set of anecdotes in which state-of-the-art models fail. Each anecdote, organised under its failure category, describes the agent's task, the hidden latent variable, the ground-truth values, trajectory-level metrics (number of turns and rewards), and the portion of the conversation that exposes the failure. The accompanying multi-turn transcript is

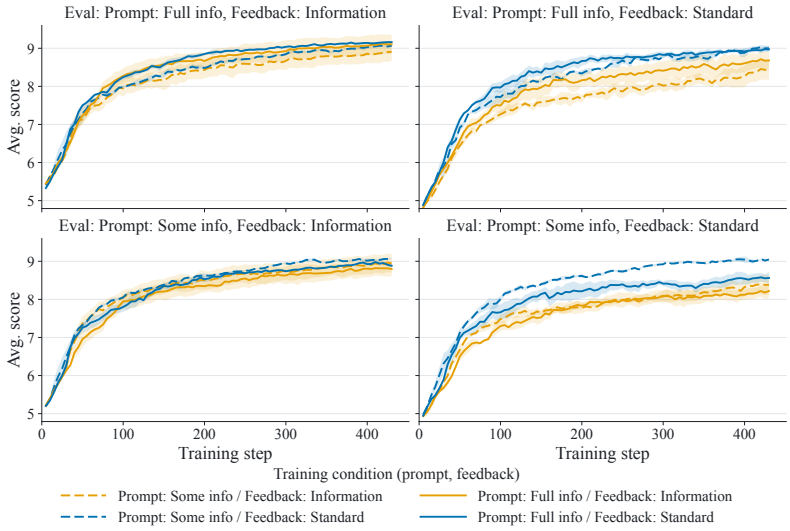


Figure 14. Roles reversed from Figure 13: each panel fixes an eval (prompt, feedback) combination and compares the four training combinations (Qwen3-8B). Same aggregation and band semantics.

colour-coded across three panels: the agent's chain of thought (when exposed) in purple, the agent's action in blue, and the game master in amber. The anecdotes cover failures across three models: GPT-4o, Gemini 2.5 Flash, and Claude Sonnet 4.6.

D.1 Adaptation Neglect

GPT-4o does not try to identify any pattern

Category: Failure to adapt across tasks
Model: GPT-4o
Configuration: Environment: number_guessing. Latent: set_of_3. Prompt: no_info. Feedback: standard. Trajectory: 0.

Setup. Targets are drawn from a fixed three-element set on every task. After 2–3 tasks a model could open with one of the previously revealed targets. GPT-4o opens tasks 1, 5 and 9 with the same midpoint guess.

What went wrong. No use of cross-task evidence; first guess is always the midpoint of the full search range.

Per-task summary (⋆ *marks tasks whose transcript is shown below*).
Agent task is to identify a hidden integer between 1 and 1000; after each guess, the game master replies whether the target is greater than, less than, or equal to the guess.
Hidden latent: set = {592, 781, 926}

Task	⋆1	2	3	⋆4	5	6	7	8	⋆9	10
Answer	781	592	926	592	926	592	926	926	592	781
Turns	5	10	8	9	8	9	8	8	9	5
Reward	0.90	0.80	0.84	0.82	0.84	0.82	0.84	0.84	0.82	0.90

Showing Task 1, Task 4 and Task 9 out of 10 tasks.

Task 1

Agent (turn 1): Let's start with a guess right in the middle to narrow it down quickly: [500].

Game Master: The number is greater than 500.

Agent (turn 2): Let's try the midpoint of the remaining range: [750].

Game Master: The number is greater than 750.

. . . continues (showing 2 of 5 turns)

Task 4

Agent (turn 1): Let's start with the middle: [500].

Game Master: The number is greater than 500.

Agent (turn 2): Let's try [750].

Game Master: The number is less than 750.

. . . continues (showing 2 of 9 turns)

Task 9

Agent (turn 1): Let's start with the middle: [500].

Game Master: The number is greater than 500.

Agent (turn 2): Let's try [750].

Game Master: The number is less than 750.

. . . continues (showing 2 of 9 turns)

Gemini does not try to identify any pattern

Category: Failure to adapt across tasks
Model: Gemini 2.5 Flash
Configuration: Environment: number_guessing. Latent: range_100. Prompt: no_info. Feedback: standard. Trajectory: 0.

Setup. Range is fixed across tasks. Gemini's openings on tasks 1, 5 and 9 do not narrow.

What went wrong. Cross-task range structure not internalized.

Per-task summary (⋆ *marks tasks whose transcript is shown below*).
Agent task is to identify a hidden integer between 1 and 1000; after each guess, the game master replies whether the target is greater than, less than, or equal to the guess.
Hidden latent: range = [1, 100]

Task	⋆1	2	3	4	⋆5	6	7	⋆8	9	10
Answer	689	630	596	680	607	682	595	675	690	672
Turns	10	9	9	7	10	9	10	10	9	6
Reward	0.80	0.82	0.82	0.86	0.80	0.82	0.80	0.80	0.82	0.88

Task 1

Agent (thinking, turn 1): ***Analyzing Game Parameters** I've started to unpack the number-guessing game's constraints. Considering a binary search... **Calculating First Move** Now, I'm setting the first guess. Based on the rules, and the initial range, I'm leaning toward 500...*

Agent (turn 1): [500]

Game Master: The number is greater than 500.

Agent (thinking, turn 2): ***Calculating Next Guess** I've narrowed the range down after the initial guess of 500.*

Agent (turn 2): [750]

Game Master: The number is less than 750.

... continues (showing 2 of 10 turns)

Task 5

Agent (thinking, turn 1): ***Initiating Game Guess***

My binary search begins anew! I'm targeting Game 5, and I'm starting from scratch with the entire range of numbers, 1 to 1000. My initial guess will be the midpoint, 500. Let's see where this takes us.

Agent (turn 1): [500]

Game Master: The number is greater than 500.

Agent (turn 2): [750]

... continues (showing 2 of 10 turns)

Task 8

Agent (thinking, turn 1): ***Initiating Game 8 Analysis***

Okay, I'm kicking things off with Game 8. My initial strategy? A binary search. To maximize efficiency right from the get-go, I'm setting my first guess at 500, the midpoint of the 1-1000 range. That's the optimal starting point for narrowing down the possibilities.

Agent (turn 1): [500]

Game Master: The number is greater than 500.

Agent (turn 2): [750]

Game Master: The number is less than 750.

... continues (showing 2 of 10 turns)

Claude does not try to identify any pattern

Category: Failure to adapt across tasks
Model: Claude Sonnet 4.6
Configuration: Environment: number_guessing. Latent: set_of_3. Prompt: no_info. Feedback: standard. Trajectory: 0.

Setup. Same three-element target set on every task. Claude opens tasks 1, 5 and 9 with the same midpoint guess and the same binary-search progression.

What went wrong. Previously revealed targets are not consulted as priors.

Per-task summary (⋆ *marks tasks whose transcript is shown below*).
Agent task is to identify a hidden integer between 1 and 1000; after each guess, the game master replies whether the target is greater than, less than, or equal to the guess.
Hidden latent: set = {592, 781, 926}

Task	⋆1	2	3	⋆4	5	6	7	8	⋆9	10
Answer	781	592	926	592	926	592	926	926	592	781
Turns	5	9	8	10	8	10	8	8	10	5
Reward	0.90	0.82	0.84	0.80	0.84	0.80	0.84	0.84	0.80	0.90

Showing Task 1, Task 4 and Task 9 out of 10 tasks.

Task 1

Agent (turn 1): I'll use a binary search strategy to find the number efficiently.

[500]

Game Master: The number is greater than 500.

Agent (thinking, turn 2): *Binary search: next guess is 750.*

Agent (turn 2): [750]

Game Master: The number is greater than 750.

. . . continues (showing 2 of 5 turns)

Task 4

Agent (thinking, turn 1): *Let me start with binary search again.*

Agent (turn 1): [500]

Game Master: The number is greater than 500.

Agent (thinking, turn 2): *[750]*

Agent (turn 2): [750]

Game Master: The number is less than 750.

. . . continues (showing 2 of 10 turns)

Task 9

Agent (thinking, turn 1): *Let me start with binary search again.*

Agent (turn 1): [500]

Game Master: The number is greater than 500.

Agent (thinking, turn 2): *[750]*

Agent (turn 2): [750]

Game Master: The number is less than 750.

. . . continues (showing 2 of 10 turns)

D.2 Adaptation Breakdown

D.2.1 Adaptation Breakdown: Models ignore the hint of hidden pattern

GPT-4o ignores the hint that there is a pattern

Configuration: Environment: number_guessing. Latent: set_of_3. Prompt: some_info. Feedback: standard. Trajectory: 0.

Setup. On set_of_3 the targets cycle through three fixed values. With `some_info`, GPT-4o is told a pattern exists. Yet the opening guess is the midpoint of the search range and the agent never tries one of the previously-seen targets first.

What went wrong. The set-of-3 hypothesis is never voiced or tested. Behavior is identical to the `no_info` regime.

Per-task summary (⋆ *marks tasks whose transcript is shown below*).
Agent task is to identify a hidden integer between 1 and 1000; after each guess, the game master replies whether the target is greater than, less than, or equal to the guess.
Hidden latent: set = {592, 781, 926}

Task	1	2	3	4	5	6	7	8	⋆9	⋆10
Answer	781	592	926	592	926	592	926	926	592	781
Turns	5	10	8	10	8	10	8	8	10	5
Reward	0.90	0.80	0.84	0.80	0.84	0.80	0.84	0.84	0.80	0.90

Showing Task 9 and Task 10 out of 10 tasks.

Task 9

Agent (turn 1): Great job so far! Let's begin Game 9 using the same effective strategy.

My first guess is: [500]

Game Master: The number is greater than 500.

Agent (turn 2): Alright, the number is greater than 500. Let's try the middle of the new range.

My next guess is: [750]

Game Master: The number is less than 750.

Agent (turn 3): Great! The number is between 501 and 749.

My next guess is: [625]

Game Master: The number is less than 625.

. . . continues (showing turns 1–7 of 10)

Task 10

Agent (turn 1): Great job so far! Let's start Game 10 with the same effective strategy.

My first guess is: [500]

Game Master: The number is greater than 500.

Agent (turn 2): Alright, the number is greater than 500. Let's try the middle of the new range.

My next guess is: [750]

Game Master: The number is greater than 750.

Agent (turn 3): Got it! The number is greater than 750. Let's narrow it down further.

My next guess is: [875]

Game Master: The number is less than 875.

. . . continues (showing turns 1–3 of 5)

D.2.2 Adaptation Breakdown: Models fail to act upon the recognized pattern

Gemini recognizes the pattern but does not act upon it

Category: Vague hints under-used
Model: Gemini 2.5 Flash
Configuration: Environment: number_guessing. Latent: dynamic_range. Prompt: some_info. Feedback: standard. Trajectory: 1.

Setup. The `some_info` hint is meant to nudge the model to look for cross-task patterns. Gemini's behavior across the trajectory does not reflect any such search.

What went wrong. No reference to the hint, no statistics over previous targets, and no narrowing of the search range across the ten tasks.

Per-task summary (⋆ *marks tasks whose transcript is shown below*).
Agent task is to identify a hidden integer between 365 and 10365; after each guess, the game master replies whether the target is greater than, less than, or equal to the guess.
Hidden latent: range covers approx. [1461, 2278]

Task	1	2	3	4	5	6	⋆7	8	9	⋆10
Answer	2233	2278	1511	1820	1983	1811	2048	1461	1958	1663
Turns	13	8	13	12	14	12	13	11	13	13
Reward	0.74	0.84	0.74	0.76	0.72	0.76	0.74	0.78	0.74	0.74

Showing Task 7 and Task 10 out of 10 tasks.

Task 7

Agent (thinking, turn 1): ***Analyzing Initial Parameters** I've begun the process by defining the initial boundaries: 365 to 10365. My strategy remains binary search. I'm calculating the midpoint to make my first guess within the provided range, aiming for an efficient approach to finding the number.
Initiating First Guess Having crunched the numbers, my first guess is in. Based on the range and my binary search strategy, I'm going with 5365. Now, I eagerly await feedback to refine my search.*

Agent (turn 1): [5365]

Game Master: The number is less than 5365.

Agent (turn 2): Okay, the number is less than 5365.

The range is now between 365 and 5364. You have 29 guesses remaining. What's your next guess? [2864]

Game Master: The number is less than 2864.

. . . continues (showing turns 1–2 of 13)

Task 10

Agent (thinking, turn 1): ***Estimating Initial Value***

I'm calculating the initial guess for the game's final round. I've determined that a binary search is the way to go, which is an efficient strategy. Starting with the given range, I need to find the midpoint. To do this, I must first calculate the range's size and then determine the midpoint using the provided formula.

Analyzing Previous Results

I've reviewed the previous nine games. The numbers, ranging from 1461 to 2278, are well below the upper bound of 10365, although I'll continue using binary search. The midpoint calculation is still correct, and my guess will remain at 5365.

Agent (turn 1): [5365]

Game Master: The number is less than 5365.

. . . continues (showing turns 1 of 13)

Claude recognizes the hint but does not act upon it strongly

Model: Claude Sonnet 4.6
Configuration: Environment: bandits. Latent: ping_pong. Prompt: some_info. Feedback: standard. Trajectory: 0.

Setup. On ping_pong with `some_info`, Claude's openings do not engage with the alternation hypothesis either — its cumulative reward (6.5–7.7) is in the same range as under `no_info`.

What went wrong. Same uniform exploration each task; no reference to prior winners.

Per-task summary (⋆ *marks tasks whose transcript is shown below*).
Agent task is to identify the button with the highest reward probability among 5 buttons (red, blue, green, yellow, purple); each turn the agent may press a button to observe a 0/1 reward, or commit via [select <button>].
Hidden latent: best button alternates: red (first) – purple (last)

Task	1	2	3	4	5	6	⋆7	8	9	⋆10
Answer	red	purple	red	purple	red	purple	red	purple	red	purple
Turns	9	10	10	15	20	11	19	8	11	11
Reward	0.86	0.85	0.85	0.00	0.70	0.83	0.00	0.88	0.83	0.83

Showing Task 7 and Task 10 out of 10 tasks.

Task 7

Agent (thinking, turn 1): *Let me review the pattern:*
- Episode 1: Red won
- Episode 2: Purple won
- Episode 3: Red won
- Episode 4: ??? (I selected green, which was wrong - might have been red or something else)
- Episode 5: Red won
- Episode 6: Purple won
Pattern seems like Red and Purple alternate or dominate. Let me start by testing red and purple, then the others.

Agent (turn 1): [red]

. . . after 17 more turns . . .

Agent (thinking, turn 19): *Updated stats: - Red: 1/3 (33%) - Blue: 0/1 (0%) - Green: 2/6 (33%) - Yellow: 3/7 (43%) - Purple: 0/1 (0%) Yellow is still leading at 43%. I've used 18 turns now. Yellow is the clear leader compared to others.*

Agent (turn 19): [select yellow]

Game Master: You selected 'yellow' as your final answer on turn 19. Wrong!

Task 10

Agent (thinking, turn 1): *Final episode! Pattern summary:*
- Episode 1: Red ✓
- Episode 2: Purple ✓
- Episode 3: Red ✓
- Episode 4: Unknown (wrong)
- Episode 5: Red ✓
- Episode 6: Purple ✓
- Episode 7: Unknown (wrong)
- Episode 8: Purple ✓
- Episode 9: Red ✓
Red and purple dominate. Let me test both quickly and focus on getting a clear winner.

Agent (turn 1): [red]

. . . after 9 more turns . . .

Agent (turn 11): Updated stats: - Red: 1/2 (50%) - Blue: 1/2 (50%) - Green: 0/1 (0%) - Yellow: 0/1 (0%) - Purple: 3/4 (75%). Purple is clearly the leader at 75%! Selecting now. [select purple]

Game Master: You selected 'purple' as your final answer on turn 11. Correct!

D.2.3 Adaptation Breakdown: Models fail to identify the pattern

Gemini tries hard but is not able to find the simple hidden pattern - biases towards that red is the favorite

Configuration: Environment: bandits. Latent: ping_pong. Prompt: some_info. Feedback: standard. Trajectory: 4.

Setup. Second Gemini trajectory on bandits/ping_pong with `some_info`. Cumulative reward 3.78 — half of GPT-4o on the same configuration. Several tasks score 0 due to commits on the wrong-side arm.

What went wrong. The agent commits aggressively on one arm and accepts the alternation tax.

Per-task summary (⋆ *marks tasks whose transcript is shown below*).
Agent task is to identify the button with the highest reward probability among 5 buttons (red, blue, green, yellow, purple); each turn the agent may press a button to observe a 0/1 reward, or commit via [select <button>].
Hidden latent: best button alternates: red (first) – purple (last)

Task	1	2	3	4	⋆5	6	7	8	9	⋆10
Answer	red	purple	red	purple	red	purple	red	purple	red	purple
Turns	26	28	26	24	24	24	24	24	24	24
Reward	0.61	0.00	0.61	0.64	0.00	0.64	0.64	0.64	0.00	0.00

Task 5

Agent (thinking, turn 1): ***Considering Game Strategies** I'm analyzing the data from Game 1, specifically the success rates for each color. Red seems promising at 61.5%, while blue is at 50%. The other colors, unfortunately, show no success. This initial data shapes my strategy for Game 5, where I'll weigh the odds based on these early insights.*
***Refining Color Probabilities** I am now focusing on the probabilities, updating them with data from Game 2 and Game 3. Red still seems decent, but its success rate dropped in Game 2. Purple showed up in Game 3 with a 36% success rate, which is interesting. I'll need to consider this in light of the larger picture. I want to calculate an overall ranking for all colors before Game 5, which seems imminent.*

Agent (turn 1): [red]

Game Master: You pressed the red button and received a reward of 0.0.

Agent (turn 2): [blue]

. . . continues (showing turns 1–2 of 24)

Task 10

Agent (thinking, turn 1): ***Analyzing Game Outcomes** Okay, I'm starting to analyze the results from the first game of ten. So far, the initial pull was heavily red, with subsequent pulls including both red and blue. I'm noting the color distribution and sequence to identify any emerging patterns within this one game, and then I'll look at the bigger picture!*
***Reviewing Game Results** I've just finished reviewing the results of the second game. The initial pulls showed red and green, and the subsequent results saw them alternating, but red's performance was not as strong this time. However, green was even weaker. I'm focusing on these fluctuations in performance to look for patterns across all games. I'll need to examine all the games and their scores to identify any significant performance trends.*

Agent (turn 1): [red]

Game Master: You pressed the red button and received a reward of 0.0.

Agent (turn 2): [blue]

. . . continues (showing turns 1–2 of 24)

Claude tries hard but fails to figure out the pattern

Configuration: Environment: secretary. Latent: threshold_06. Prompt: some_info. Feedback: standard. Trajectory: 1.

What went wrong. Claude fails to recognize the parttern.

Per-task summary (⋆ *marks tasks whose transcript is shown below*).
Agent task is to pick the maximum of 10 values revealed one at a time; on each turn the agent chooses [accept] (claim the current value and end the task) or [continue] (skip).
Hidden latent: max value > 0.6; all others < 0.6

Task	1	2	3	4	5	6	7	8	⋆9	⋆10
Answer	0.69 at 6	0.72 at 3	0.69 at 9	0.63 at 8	0.73 at 8	0.72 at 10	0.62 at 6	0.63 at 7	0.70 at 1	0.73 at 9
Turns	9	11	8	9	9	7	6	6	11	7
Reward	1.00	0.33	0.38	1.00	1.00	0.31	1.00	0.42	0.13	0.37

Task 9

Agent (thinking, turn 1): *Let me think about the pattern of where the maximum has appeared in previous rounds:*

Episode 1: Max was at position 6 (0.6937) - accepted correctly
Episode 2: Max was at position 3 (0.7158) - missed it, ended with 0.4755
Episode 3: Max was somewhere after position 7 (0.5239 wasn't the max)
Episode 4: Max was at position 8 (0.6250) - accepted correctly
Episode 5: Max was at position 8 (0.7349) - accepted correctly
Episode 6: Max was somewhere after position 6 (0.4481 wasn't the max)
Episode 7: Max was at position 6 (0.6178) - accepted correctly
Episode 8: Max was somewhere after position 6 (0.5276 wasn't the max)
Now I'm at position 1 with a value of 0.7004, which is quite strong compared to the historical maximums I've seen. Following the secretary problem's optimal stopping rule would suggest skipping the first ˜37% of candidates, but this current value is already in the upper range of what I've encountered before, so I need to weigh whether to accept it now or continue searching.

Agent (turn 1): [continue]

. . . continues (showing turn 1 of 11)

Task 10

Agent (thinking, turn 1): *Let me think about the patterns I've observed across the games:*

Game 1: Max was at position 6 (0.6937) - I accepted it ✓
Game 2: Max was at position 3 (0.7158) - I missed it, ended up accepting position 10 ×
Game 3: Max was somewhere after position 7 - I accepted 0.5239 at position 7 ×
Game 4: Max was at position 8 (0.6250) - I accepted it ✓
Game 5: Max was at position 8 (0.7349) - I accepted it ✓
Game 6: Max was somewhere after position 6 - I accepted 0.4481 at position 6 ×
Game 7: Max was at position 6 (0.6178) - I accepted it ✓
Game 8: Max was somewhere after position 6 - I accepted 0.5276 at position 6 ×
Game 9: Max was at position 1 (0.7004) - I missed it ×
Looking at where the maximum actually appeared—6, 3, unknown, 8, 8, unknown, 6, unknown, 1—I'm noticing the positions vary quite a bit, and I've been stopping too early in several cases. This is the final game, so I need to be more strategic about when to accept versus continue searching.
My approach is to skip the first few positions and then take the first value that exceeds everything I've seen so far. Right now at position 1 with 0.5315, I'm still in the observation phase, so I'll keep looking.

Agent (turn 1): [continue]

. . . continues (showing turns 1 of 7)

D.2.4 Adaptation Breakdown: Model tries too hard to fit a pattern leading to suboptimal behavior

Gemini trying hard to find the pattern - finds that it was wrong - abondons the binary search on the given task. And once failed on a task doent even attempt to find the pattern.

Configuration: Environment: number_guessing. Latent: two_ranges. Prompt: some_info. Feedback: standard. Trajectory: 0.

Setup. On task 5 of two_ranges, Gemini begins with binary search (3750, 4375, ...) but after crossing the midpoint switches to a '+1 each turn' hypothesis and increments linearly, exhausting all 30 turns without finding 5900. Episode reward: 0; trajectory cumulative: 6.74.

What went wrong. The model commits to a wrong incrementing strategy after binary search — once committed, it does not revisit the binary-search hypothesis and wastes turns.

Per-task summary (⋆ *marks tasks whose transcript is shown below*).
Agent task is to identify a hidden integer between 1 and 10000; after each guess, the game master replies whether the target is greater than, less than, or equal to the guess.
Hidden latent: targets drawn from two disjoint ranges

Task	**1**	**2**	**3**	**4**	**⋆5**	**6**	**7**	**8**	**9**	**⋆10**
Answer	2721	2729	2702	2501	5900	2715	6233	2805	5821	2844
Turns	12	11	13	12	30	13	13	13	13	13
Reward	0.76	0.78	0.74	0.76	0.00	0.74	0.74	0.74	0.74	0.74

Transcript

Task 5

Agent (thinking, turn 1): *(strategy at start of episode) **Analyzing Game Patterns** I've just finished analyzing the first four game rounds. The initial number guesses were 2721 (12 turns), 2729 (11 turns), 2702 (13 turns), and 2501 (12 turns). All guesses have fallen between 2501 and 2729. I am beginning to look at if there's any pattern there.*

***Refining Initial Guesses** I've got a clearer picture of the number range: [2501, 2729]. The average is around 2663.25. While binary search is generally optimal, I'm now weighing the risk/reward of deviating. Should I aim for the middle of this observed sub-range (maybe 2600-2700) to speed up if the pattern continues, or stick with a binary search-type approach given the "might" in "might follow a pattern" condition? Considering max turns are 30, and the initial range of 10000.*

***Re-evaluating Optimal Strategy** I'm now zeroing in on an optimal initial guess, based on the accumulated data...*

... first 8 turns of this task omitted

Agent (turn 9): [4995]

Game Master: The number is greater than 4995.

Agent (turn 10): [4998]

Game Master: The number is greater than 4998.

Agent (turn 11): [4999]

Game Master: The number is greater than 4999.

Agent (turn 12): [5000]

Game Master: The number is greater than 5000.

Agent (turn 13): [5001]

Game Master: The number is greater than 5001.

Agent (turn 14): [5002]

... continues (showing turns 9–14 of 30)

Task 10

Agent (thinking, turn 1): ***Analyzing Game Trends***

I'm now starting Game 10. Based on the data from the previous nine games, I'm focusing on the number of turns and final scores. The scores are consistently low, under 5000. Each game is between 11-13 turns. I'll make my initial guess soon, keeping these trends in mind.

Evaluating Recent Results

I've just finished reviewing the outcomes from Games 1-9. I'm seeing a trend: most scores cluster around 2500-3000, but there are also a couple of higher scores, in the...

Agent (turn 1): I'm thinking of a number between 1 and 10000. You have 30 guesses. What's your guess? [5000]

Game Master: The number is less than 5000.

Agent (turn 2): [2500]

Game Master: The number is greater than 2500.

Agent (turn 3): [3750]

Game Master: The number is less than 3750.

Agent (turn 4): [3125]

Game Master: The number is less than 3125.

Agent (turn 5): [2812]

... continues (showing turns 1–7 of 13)

Claude wrongly reasons about a failed task and tries too hard to reason for the pattern

Model: Claude Sonnet 4.6
Configuration: Environment: bandits. Latent: ping_pong. Prompt: some_info. Feedback: standard. Trajectory: 4.

Setup. On ping_pong with `some_info`, Claude's trajectory 4 cumulative reward is 4.76 — well below its best ping_pong trajectory (7.71). Several tasks score 0 because the agent commits to one arm and the latent alternates the answer.

What went wrong. After a couple of wins on one arm, Claude commits and stops rotating. The alternation latent then punishes the very next task. Reasoning is verbose; the policy is brittle.

Per-task summary (⋆ *marks tasks whose transcript is shown below*).
Agent task is to identify the button with the highest reward probability among 5 buttons (red, blue, green, yellow, purple); each turn the agent may press a button to observe a 0/1 reward, or commit via [select <button>].
Hidden latent: best button alternates: red (first) – purple (last)

Task	1	2	3	4	5	6	⋆7	8	9	⋆10
Answer	red	purple	red	purple	red	purple	red	purple	red	purple
Turns	8	11	9	15	18	15	14	16	12	18
Reward	0.88	0.83	0.00	0.78	0.73	0.78	0.00	0.76	0.00	0.00

Showing Task 7 and Task 10 out of 10 tasks.

Task 7

Agent (thinking, turn 1): *Pattern so far: Game 1=Red, Game 2=Purple, Game 3=Green, Game 4=Purple, Game 5=Red, Game 6=Purple.*
Purple has won 3 times, Red 2 times, Green 1 time.
Blue and Yellow haven't won yet.
Let me explore all buttons systematically again.

Agent (turn 1): [red]

Game Master: You pressed the red button and received a reward of 0.0.

Agent (thinking, turn 2): *Let me explore all buttons to get initial data.*

Agent (turn 2): [blue]

. . . continues (showing turns 1–2 of 14)

Task 10

Agent (thinking, turn 1): *This is very frustrating. I've been getting unlucky with buttons that appear to have high probabilities but aren't the actual highest. The issue seems to be that multiple buttons can have decent probabilities, and I need more samples to differentiate. Let me think about what I know: - Game 1: Red won - Game 2: Purple won - Game 3: Green won (I picked yellow with 3/3 and was wrong) - Game 4: Purple won - Game 5: Red won - Game 6: Purple won - Game 7: Unknown winner (I picked yellow with 2/5,. . .*

Agent (turn 1): This is the last game. I've noticed that even buttons with 4/4 results can be wrong - there might be a button with an even higher probability. I need to be very thorough this time. Let me test each button multiple times. [red]

Game Master: You pressed the red button and received a reward of 0.0.

. . . continues (showing turns 1 of 18)

D.3 Adaptation Miscalibration

GPT 4o does not know how to use the latent information

Category: Full_info confusion
Model: GPT-4o
Configuration: Environment: number_guessing. Latent: two_ranges. Prompt: full_info. Feedback: standard. Trajectory: 0.

Setup. The `full_info` prompt describes the disjoint-ranges structure verbosely. GPT-4o under `full_info` scores below its own `some_info` performance — tasks 8 and 9 both end at zero reward despite GPT-4o correctly hypothesising both candidate ranges at turn 1.

What went wrong. GPT-4o opens both episodes with the right two-range hypothesis (∼2501–3000 and ∼5900–6399), but after the first feedback it commits to a single range and exhausts the 30-turn budget there. By the time it crosses to the other range the budget is already gone. Task 8 (target 2805, lower range) is searched in the upper range until turn 27; Task 9 (target 5821, upper range) is searched in the lower range until turn 26.

Per-task summary (⋆ *marks tasks whose transcript is shown below*).
Agent task is to identify a hidden integer between 1 and 10000; after each guess, the game master replies whether the target is greater than, less than, or equal to the guess.
Hidden latent: targets drawn from two disjoint ranges

Task	1	2	3	4	5	6	7	⋆8	⋆9	10
Answer	2721	2729	2702	2501	5900	2715	6233	2805	5821	2844
Turns	12	12	15	10	11	9	9	30	30	5
Reward	0.76	0.76	0.70	0.80	0.78	0.82	0.82	0.00	0.00	0.90

Task 8

Agent (turn 1): Based on previous games, we've identified two potential ranges: 2501 to 3000 and 5900 to 6399. Let's start by guessing the middle of these ranges to quickly determine which range the number might fall into: [2750]

Game Master: The number is greater than 2750.

. . . after 28 more turns . . .

Intermediate guesses (turns 2–29): `6150, 6025, 5962, 5930, 5915, 5907, 5903, 5901, 5900, 5899, 5895, 5880, 5850, 5800, 5700, 5600, 5500, 5400, 5300, 5200, 5100, 5000, 4500, 4250, 4000, 3500, 3250, 3000`

Agent (turn 30): The number is less than 3000. Let's narrow it down further within the known range: [2750]

Game Master: The number is greater than 2750.
You've run out of guesses.

Task 9

Agent (turn 1): Based on previous games, we've identified two potential ranges: 2501 to 3000 and 5900 to 6399. Let's start with the middle of these ranges to quickly determine which range the number might fall into: [2750]

Game Master: The number is greater than 2750.

. . . after 28 more turns . . .

Intermediate guesses (turns 2–28): `2875, 2938, 2970, 2985, 2993, 2997, 2999, 3000, 6200, 2750, 2875, 2938, 2970, 2985, 2993, 2997, 2999, 3000, 3100, 3500, 4000, 5000, 5500, 6000, 5750, 5875, 5812`

Agent (turn 30): The number is between 5813 and 5874. Let's narrow it down further: [5843]

Game Master: The number is less than 5843.
You've run out of guesses.

GPT-4o full info underperforms as compared to som info on same trajectory (some=5.93 vs full=2.21)

Category: Full_info confusion
Model: GPT-4o
Configuration: Environment: secretary. Latent: increasing_position. Prompt and feedback: see sub-sections below. Trajectory: 4.

Setup. On increasing_position the optimal accept-position grows by one each task. Under `some_info` GPT-4o scores 5.93 cumulative; under `full_info` on the same trajectory it scores only 2.21. The verbose description of the latent does not produce an incrementing accept policy — instead it triggers premature accepts on early indices.

What went wrong. Full_info names the latent in plain English yet GPT-4o does not implement the corresponding policy. Each task is solved with the same heuristic, often accepting too early; cumulative reward is roughly half the some_info baseline.

Per-task summary (⋆ *marks tasks whose transcript is shown below*).

full_info
Agent task is to pick the maximum of 10 values revealed one at a time; on each turn the agent chooses [accept] (claim the current value and end the task) or [continue] (skip and see the next).
Hidden latent: max position grows by 1 each task (task k -> position k)

Task	**⋆1**	**2**	**3**	**4**	**5**	**⋆6**	**7**	**8**	**⋆9**	**10**
Answer	1.00 at 1	1.00 at 2	0.88 at 3	0.97 at 4	0.81 at 5	0.96 at 6	1.00 at 7	0.92 at 8	1.00 at 9	1.00 at 10
Turns	8	8	8	8	4	4	4	4	4	4
Reward	0.47	0.47	0.13	0.16	0.44	0.09	0.29	0.07	0.07	0.03

some_info
Agent task is to pick the maximum of 10 values revealed one at a time; on each turn the agent chooses [accept] (claim the current value and end the task) or [continue] (skip and see the next).
Hidden latent: max position grows by 1 each task (task k -> position k)

Task	**⋆1**	**2**	**3**	**4**	**5**	**⋆6**	**7**	**8**	**⋆9**	**10**
Answer	1.00 at 1	1.00 at 2	0.88 at 3	0.97 at 4	0.81 at 5	0.96 at 6	1.00 at 7	0.92 at 8	1.00 at 9	1.00 at 10
Turns	8	2	10	4	10	6	5	10	7	2
Reward	0.47	1.00	0.44	1.00	0.23	1.00	0.47	0.38	0.49	0.45

Gemini on wordladder/hub_word_4letter, trajectory 2: full_info collapses (some=4.85 vs full=2.63) - tries to go for the latent without being confident

Category: Full_info confusion
Model: Gemini 2.5 Flash
Configuration: Environment: wordladder. Latent: hub_word_4letter. Prompt and feedback: see sub-sections below. Trajectory: 2.

Setup. On the same trajectory 2, Gemini's cumulative reward drops from 4.85 (some_info) to 2.63 (full_info). Per-task rewards under full_info: `[0.91, 0.50, 0.25, 0.97, 0.00]`. The verbose hub-word framing destabilizes word selection.

What went wrong. Gemini under full_info submits words that fit the hub-word narrative but break the one-letter-change rule, accumulates invalid moves, and ends task 5 at zero reward.

Per-task summary (⋆ *marks tasks whose transcript is shown below*).

full_info
Agent task is to transform a given start word into a given target word (4-letter words) by changing exactly one letter per turn; every intermediate word must be a valid English word.
Hidden latent: all optimal paths pass through hub word = cork

Task	**⋆1**	**2**	**⋆3**	**4**	**⋆5**
Answer	cord to farm	care to lock	boot to care	hand to work	boat to worm
Turns	7	21	21	6	21
Reward	0.91	0.50	0.25	0.97	0.00

some_info
Agent task is to transform a given start word into a given target word (4-letter words) by changing exactly one letter per turn; every intermediate word must be a valid English word.
Hidden latent: all optimal paths pass through hub word = cork

Task	**⋆1**	**2**	**⋆3**	**4**	**⋆5**
Answer	cord to farm	care to lock	boot to care	hand to work	boat to worm
Turns	4	8	9	4	5
Reward	1.00	0.94	0.91	1.00	1.00

Full info

Showing Task 1, Task 3 and Task 5 out of 5 tasks.

Task 1

Agent (thinking, turn 1): ***Discovering the Hub Word***

I am now focusing on finding the "hub" word that links "cord" to "farm". The user wants me to discover it for the game. I think I am getting closer to the solution; if I can identify the pattern, this should be easier.

Testing the Hub Hypothesis

I am now testing whether "form" functions as the hub word for this word ladder. The path is direct: "cord" to "corn" to "form," and then "form" to "farm." It seems quite efficient, at only three steps, which confirms its suitability as a potential intermediary. Now, I need to check how it will function within the broader sequence.

Agent (turn 1): [corn]

Game Master: Nice! Keep going.
Word Ladder History: cord -> corn. Target Word: farm

... continues (showing turns 1–1 of 7)

Task 3

Agent (thinking, turn 1): ***Mapping the Path***

I'm working on the word ladder, going from "boot" to "care" in Game 3. Starting with 'boot' and aiming for 'care', I'm planning my moves one letter at a time, aiming for optimal steps. The "boot" hint is guiding my choices!

Identifying the Hub Word

I'm now focusing on identifying the intermediate "hub" word, drawing on previous games for clues. Game 1's path, 'cord -> care -> farm', suggests 'care'. Game 2 has me starting at 'care' to reach 'lock'. That feels like my path may have a starting point and an ending point. I now suspect 'care' serves as the hub, and I'm developing a strategy to use this assumption in Game 3.

Agent (turn 1): [boar]

Game Master: You attempted an invalid move. Reason: 'boar' is not exactly one letter different from 'boot'.

... continues (showing turns 1–1 of 21)

Task 5

Agent (thinking, turn 1): ***Considering the Constraints***

I'm focusing on the Word Ladder problem, specifically the 'boat' to 'worm' challenge. I'm reminded that solutions often converge at a crucial "hub" word. I'm actively seeking this hub, as the constraints dictate a one-letter change per step progression. This hub strategy will be central to resolving the final game in the series.

Analyzing Past Word Ladders

I'm now diving into the previous games to spot potential "hub" words. While 'care' appeared in the initial game and as a target, I don't feel confident that it's a central hub word. It seems I didn't identify a clear hub in the initial word ladder. I'm reviewing my attempt to find some recurring themes in order to create the series-ending path.

Agent (turn 1): [boar]

Game Master: Nice! Keep going.
Word Ladder History: boat -> boar. Target Word: worm

... continues (showing turns 1–1 of 21)

Some Info

Showing Task 1, Task 3 and Task 5 out of 5 tasks.

Task 1

Agent (thinking, turn 1): ***Initiating Word Transformation***

I'm working on the Word Ladder game now. Starting with 'cord', I'm aiming for 'farm'. Considering the rules, my first step is to get closer by changing one letter. I've considered some options, and I think I'll change the first letter.

Calculating Optimal Paths

I've evaluated several paths, focusing on single-letter changes to move from 'cord' to 'farm'. I've identified the shortest route: CORD, FORD, FORM, FARM, and validated it. The key was prioritizing valid English words with each transformation. I'm satisfied with this solution.

Agent (turn 1): [ford]

Game Master: Nice! Keep going.
Word Ladder History: cord -> ford. Target Word: farm

... continues (showing turns 1–1 of 4)

Task 3

Agent (thinking, turn 1): ***Mapping the Path***

I've begun mapping a word path from "boot" to "care," focusing on single-letter changes. My current path looks like this: boot -> coot. I'm actively brainstorming the next word to progress towards the target "care", while ensuring the changes are valid English words.

Refining the Word Path

I've discarded the initial, lengthy path and am now focusing on "boot -> moot -> root -> soot -> shot -> chat -> cart -> care." The strategy has shifted towards minimizing steps. The current path is eight steps, which looks promising. Now, I'm analyzing it for potential shortcuts and optimizations, keeping an eye out for ways to reach "care" more directly.

Agent (turn 1): [moot]

Game Master: Nice! Keep going.
Word Ladder History: boot -> moot. Target Word: care

... continues (showing turns 1–1 of 9)

Task 5

Agent (thinking, turn 1): ***Charting a Path Forward***

I've got the start word "boat" and the end word "worm". So far, I've got a good start: boat -> moat. Now, I'm thinking, what's the next logical step to get closer to "worm"? I'm looking for a valid word that changes only one letter from "moat". I'm considering all possibilities.

Refining the Approach

I'm back to the initial step, boat to moat. Now, I'm stuck again! I'm struggling to get from "moat" to "worm" in single-letter steps. I need a word between "moat" and "worm". I'm rethinking my strategy. Maybe I should consider other words from "boat."

Agent (turn 1): [moat]

Game Master: Nice! Keep going.
Word Ladder History: boat -> moat. Target Word: worm

... continues (showing turns 1–1 of 5)

Claude on bandits/ping_pong, trajectory 0: full_info hurts (some=6.65 vs full=4.31)

Category: Full_info confusion
Model: Claude Sonnet 4.6
Configuration: Environment: bandits. Latent: ping_pong. Prompt and feedback: see subsections below. Trajectory: 0.

Setup. Claude on `traj_0000`: cumulative reward drops from 6.65 (some_info) to 4.31 (full_info). The full description of the alternation latent leads Claude to commit to a single arm rather than rotate.

What went wrong. Verbose alternation reasoning translates into over-commitment, not into a rotating policy. Cumulative reward is well below the some_info baseline.

Per-task summary (⋆ *marks tasks whose transcript is shown below*).

full_info
Agent task is to identify the button with the highest reward probability among 5 buttons (red, blue, green, yellow, purple); each turn the agent may press a button to observe a 0/1 reward, or commit via [select <button>].
Hidden latent: best button alternates: red (first) – purple (last)

Task	**⋆1**	**2**	**3**	**4**	**5**	**⋆6**	**7**	**⋆8**	**9**	**10**
Answer	red	purple	red	purple	red	purple	red	purple	red	purple
Turns	9	17	10	12	10	11	14	12	4	13
Reward	0.86	0.00	0.85	0.00	0.85	0.00	0.00	0.00	0.94	0.80

some_info
Agent task is to identify the button with the highest reward probability among 5 buttons (red, blue, green, yellow, purple); each turn the agent may press a button to observe a 0/1 reward, or commit via [select <button>].
Hidden latent: best button alternates: red (first) – purple (last)

Task	**⋆1**	**2**	**3**	**4**	**5**	**⋆6**	**7**	**⋆8**	**9**	**10**
Answer	red	purple	red	purple	red	purple	red	purple	red	purple
Turns	9	10	10	15	20	11	19	8	11	11
Reward	0.86	0.85	0.85	0.00	0.70	0.83	0.00	0.88	0.83	0.83

Full Info

Task 1

Agent (turn 1): I'll start by exploring each button to gather initial data, then focus on the most promising ones.

Let me explore all 5 buttons first to get a baseline.

[red]

Game Master: You pressed the red button and received a reward of 1.0.

Agent (thinking, turn 2): *Good start! Red gave a reward. Let me explore the others.*

Agent (turn 2): Good start with red! Let me try the others.

[blue]

Game Master: You pressed the blue button and received a reward of 1.0.

. . . continues (showing turns 1–2 of 9)

Task 6

Agent (thinking, turn 1): *Pattern so far:*
- Game 1: Red ✓
- Game 2: not yellow
- Game 3: Red ✓
- Game 4: not green
- Game 5: Red ✓
Interesting - odd games (1, 3, 5) are all red. Even games (2, 4) are unknown. Game 6 is even. Let me think about what the pattern might be for even games. Game 2: not yellow Game 4: not green Game 6: maybe blue or purple? The hint says "The best color changes in a predictable pattern across games." Let me think about this differently. Maybe the pattern cycles through colors in order: red, blue, green, yellow, purple, red, blue, green, yellow, purple... Wait, that doesn't work since red keeps appearing in odd positions. Let me reconsider - maybe odd games are always red, and even games cycle through the other colors. So game 2 could be blue, green, or purple (not yellow), game 4 could be blue, yellow, or purple (not green), and game 6 would be one of the remaining options. I should focus on testing blue and purple for game 6 since those seem most likely. I'm realizing I don't have enough data to pin down the pattern yet, so I should just test the remaining options in game 6 systematically—starting with blue and purple since those are still viable candidates.

. . . after 9 more turns . . .

Agent (turn 11): Excellent! Green is 4/4 (100%)! Very compelling evidence. Updated scores: - Red: 0/1 (0%) - Blue: 0/1 (0%) - Green: 4/4 (100%) - Yellow: 0/1 (0%) - Purple: 2/3 (67%) Green is clearly dominant here! [select green]

Game Master: Wrong!

Task 8

Agent (thinking, turn 1): *Let me reconsider the pattern. Looking at what I know:*
- Game 1: Red ✓ - Game 2: Not yellow - Game 3: Red ✓ - Game 4: Not green - Game 5: Red ✓ - Game 6: Not green - Game 7: Not blue Wait, the problem says "The best color changes in a predictable pattern across games." But I've been wrong about the even games. Let me think harder The odd games (1, 3, 5) were RED. But game 7 was also wrong, which breaks my assumption that odd games are always red. Maybe the pattern cycles through a set of colors rather than alternating between two. If it's something like red, blue, green, yellow repeating, then game 1 would be red, game 2 blue, game 3 green—but that doesn't match what I know either. I need to reconsider whether red was actually correct for games 1, 3, and 5, or if I just happened to guess right those times. I'm realizing game 7 breaks the odd-equals-red pattern since I guessed blue and was wrong. Let me test a simpler alternating hypothesis: red, blue, red, blue, red, blue, red, blue. ..... OK so possible patterns: 1. Alternating Red/Blue: R, B, R, B, R, B, R, B, R, B -> Game 8: Blue 2. All odd = Red ..... I'm confident in the even/odd pattern I've identified, so I'll commit to Blue for game 8.

Agent (turn 1): [blue]

. . . after 10 more turns . . .

Agent (turn 12): Updated scores: - Red: 1/2 (50%) - Blue: 1/2 (50%) - Green: 0/1 (0%) - Yellow: 3/4 (75%) - Purple: 1/2 (50%). Yellow is clearly the strongest at 75% with 4 samples! [select yellow]

Game Master: Wrong!

Some Info

Task 1

Agent (turn 1): I'll start by exploring each button to get initial data, then focus on the most promising ones.

[red]

Game Master: You pressed the red button and received a reward of 1.0.

Agent (thinking, turn 2): *Good start! Red gave a reward. Let me explore the other buttons.*

Agent (turn 2): Good start! Red gave a reward. Let me explore the others.

[blue]

Game Master: You pressed the blue button and received a reward of 1.0.

. . . continues (showing turns 1–2 of 9)

Task 6

Agent (thinking, turn 1): *Let me review what I've learned so far: - Episode 1: Red won - Episode 2: Purple won - Episode 3: Red won - Episode 4: Wrong (selected green, likely red was correct) - Episode 5: Red won. Red seems to be the dominant button. Let me start with red and see if I can confirm quickly.*

Agent (turn 1): Looking at the pattern: Red has won episodes 1, 3, 5 - it seems to be the dominant button. Let me start with red.

[red]

Game Master: You pressed the red button and received a reward of 0.0.

Agent (thinking, turn 2): *Red started with a loss this time. Let me explore the others.*

Agent (turn 2): Red missed this time. Let me explore the others.

[blue]

Game Master: You pressed the blue button and received a reward of 0.0.

. . . continues (showing turns 1–2 of 11)

Task 8

Agent (thinking, turn 1): *Let me think about the pattern more carefully. The pattern seems to be: Episode 1: Red*
Episode 2: Purple
Episode 3: Red
Episode 4: ? (not green)
Episode 5: Red
Episode 6: Purple
Episode 7: ? (not yellow)
Maybe the pattern alternates between two buttons, or maybe there's a specific sequence. Looking at the odd episodes, they're consistently red, while the even episodes I've confirmed are purple. But episodes 4 and 7 don't fit this simple alternation. Maybe blue works for episode 4, and green for episode 7 since it had better odds than yellow. I should approach this more systematically instead of guessing—the winning button likely has a very high success rate that I'm just missing when I don't test it early enough. For episode 8, I'll explore more carefully and only commit to a button once I have solid evidence like 4+ successes, rather than selecting when the data is still ambiguous.

Agent (turn 1): Let me explore carefully. [red]

. . . after 6 more turns . . .

Agent (thinking, turn 8): *Purple is 3/3 (100%)! I'm very confident. Let me select purple now.*

Agent (turn 8): [select purple]

Game Master: You selected 'purple' as your final answer on turn 8. Correct!