# SAMPLE-EFFICIENT LEARNING FROM AGENT EXPERIENCE

**Chenhui Gou**[1,2,*] **Haoqin Tu**[2,§] **Yunhao Fang**[2,§] **Jianfei Cai**[1] **Hamid Rezatofighi**[1]
[1]Monash University [2]ByteDance Seed



## ABSTRACT

Real-world agent learning is often constrained by costly environment interactions, such as running time-consuming experiments or obtaining human feedback. In-context learning offers a highly sample-efficient way for agents to learn from their own interaction histories, but its gains disappear once that experience is removed from the context. Separately, context distillation provides a mechanism for internalizing contextual information into model weights. However, applying it to agents' interaction histories without sacrificing environment sample efficiency remains underexplored. We term this problem Experience Distillation and develop an implementation that requires no further environment interaction beyond the collected experience. Experiments on 749 curated software-engineering tasks and six text-adventure games show that it retains at least 64.8% of the gains from in-context learning across both domains, whereas direct supervised fine-tuning on the collected experience recovers only 3.8%. Compared with classical reinforcement-learning baselines, in-context learning from trial-and-error experience followed by Experience Distillation matches their performance with at least $9.6\times$ fewer environment samples.

## 1 INTRODUCTION

Real-world tasks can demand substantial time, resources, or human expertise. Testing a proposed battery material may require laboratory synthesis and measurement, whereas evaluating a legal analysis may require review by a qualified lawyer. As agents become capable of handling longer and more realistic tasks (METR, 2026; Kwa et al., 2025), collecting rollouts from the environment becomes increasingly costly. Reinforcement learning (RL) is a standard framework for learning through interaction with an environment. However, it often has poor environment sample efficiency, requiring large numbers of environment interactions to achieve strong performance (Haarnoja et al., 2018; Kaiser et al., 2020; Narvekar et al., 2020). This demand is manageable when environment interaction is cheap and scalable, such as in Go self-play (Silver et al., 2016) or lightweight coding tasks (Guo et al., 2025). When environment interaction is costly, the same sample demand can make training impractical.

In-context learning (ICL) offers a sample-efficient way for large language models to adapt using task-specific evidence in the context. By conditioning on a few demonstrations or on feedback from previous attempts, ICL can improve an agent's behavior without requiring a large training set or extensive environment interaction (Brown et al., 2020; Garg et al., 2022; Laskin et al., 2023; Zhu et al., 2026). However, ICL does not update the model parameters, so the improvement disappears once the context is removed. Separately, context distillation provides a natural mechanism for consolidating these gains into model weights. It trains a student without informative context to reproduce the behavior of a teacher conditioned on that context (Hinton et al., 2015; Askell et al., 2021; Snell et al., 2022). Combining ICL with context distillation therefore appears to offer a sample-efficient way to learn through interaction with an environment (Duan et al., 2024). Recent work has begun to explore this idea in agent tasks (Ye et al., 2026a;b; Penaloza et al., 2026; Wang et al., 2026). However, distilling an experience-conditioned teacher requires running it in the environment again. A practical, sample-efficient method should therefore minimize additional environment interaction during distillation. We term this problem **Experience Distillation**.

[*]Correspondence: Chenhui.Gou@monash.edu.
[§]Work done while at ByteDance Seed.

To avoid further environment interaction, a naive solution inspired by model-based RL is to run the experience-conditioned teacher in a learned world model rather than the real environment (Sutton & Barto, 2018; Chua et al., 2018). However, even small errors from the world model build up over long rollouts, making the generated trajectories unreliable (Talvitie, 2017). Branched rollouts reduce this problem by starting from points sampled along collected trajectories and rolling out in the world model for only a few steps (Janner et al., 2019). By reducing each branch to a single step and stopping after the teacher's action, we obtain a practical implementation of Experience Distillation that requires neither a world model nor further environment interaction.

We evaluate this procedure on 749 curated software-engineering tasks and six text-adventure games. In our experiments, a task's experience context typically contains roughly 60–600 interaction turns and over 80k tokens. A single multi-task distillation run contains up to 61.7M experience tokens in aggregate. After Experience Distillation, the models retain at least 64.8% of the performance gains from ICL, compared with only 3.8% after direct SFT on the same experience. The combination of trial-and-error ICL and Experience Distillation also matches classical RL baselines while using at least $9.6\times$ fewer environment samples. We hope these initial results motivate further research on Experience Distillation and, more broadly, sample-efficient learning from agent experience.

## 2 Preliminaries

**Context Distillation.** Context distillation is a form of knowledge distillation in which the teacher receives informative context unavailable to the student (Hinton et al., 2015; Askell et al., 2021; Snell et al., 2022). Let $x$ denote an input and $c$ an auxiliary context, such as a demonstration, document, memory, or other privileged information (Charakorn et al., 2026; Zheng et al., 2026; Penaloza et al., 2026). A teacher with parameters $\theta_0$ induces $p_{\theta_0}(y \mid x, c)$, while a student with parameters $\theta$ observes only $x$. For a fixed $(x, c)$, context distillation minimizes

$$\mathcal{L}_{\mathrm{CD}}(\theta; x, c) = D_{\mathrm{KL}}(p_{\theta_0}(\cdot \mid x, c) \,\|\, p_\theta(\cdot \mid x)) \,.$$

**Interactive Agent Tasks.** We model each interactive agent task as a partially observable Markov decision process (POMDP) $M$ (Kaelbling et al., 1998; Sutton & Barto, 2018). At step $t$, the agent receives an observation $o_t$ and selects an action $a_t$ based on the observable history $h_t = (o_0, a_0, \ldots, o_t)$. Following the percept view in general reinforcement learning (Veness et al., 2010), we treat rewards and other environment feedback as part of $o_t$. Data collection produces a finite observable trajectory prefix $\tau = (o_0, a_0, \ldots, a_{T-1}, o_T)$, where $T$ denotes the length of the collected prefix. When a model with parameters $\theta$ serves as the policy $\pi_\theta$, its interaction with $M$ induces the trajectory-prefix distribution

$$p_M^\theta(\tau) = p_M(o_0) \prod_{t=0}^{T-1} \pi_\theta(a_t \mid h_t)\, p_M(o_{t+1} \mid h_t, a_t).$$

Here, $p_M(o_0)$ is the initial-observation distribution and $p_M(o_{t+1} \mid h_t, a_t)$ is the history-conditioned next-observation distribution; both are induced by $M$ after marginalizing over its latent states.

## 3 Experience Distillation

Our goal is to distill collected experience into model weights while preserving environment sample efficiency. We formulate this goal as trajectory-level context distillation and derive an implementation based on one-step branched rollouts that requires neither additional environment interaction nor a world model.

### 3.1 Naive Context Distillation from Experience

Let $M \sim \mathcal{D}$ be a task and $\theta_0$ the base model parameters. Using the notation of Section 2, we denote experience collected by the base model as $\tau^{\mathrm{exp}} \sim p_M^{\theta_0}(\cdot)$. When the base model attempts the task again with $\tau^{\mathrm{exp}}$ in context, it induces a distribution over new trajectories $\tau'$, written $p_M^{\theta_0}(\tau' \mid \tau^{\mathrm{exp}})$. Running the student on the same task without the experience context instead induces $p_M^{\theta}(\tau')$. For a

given $M$ and $\tau^{\text{exp}}$, the ideal context distillation objective is

$$\begin{aligned}\mathcal{L}_{\text{CD}}(\theta; M, \tau^{\text{exp}}) &= D_{\text{KL}}\left(p_M^{\theta_0}(\tau' \mid \tau^{\text{exp}}) \,\|\, p_M^{\theta}(\tau')\right) \\ &= \mathrm{E}_{\tau' \sim p_M^{\theta_0}(\cdot \mid \tau^{\text{exp}})}\left[\sum_t D_{\text{KL}}\left(\pi_{\theta_0}(\cdot \mid h_t, \tau^{\text{exp}}) \,\|\, \pi_\theta(\cdot \mid h_t)\right)\right].\end{aligned} \tag{1}$$

Using the trajectory factorization in Section 2, the teacher and student distributions share the same initial-observation and environment-response factors, while their policy factors differ. Applying the chain rule for KL therefore yields the second equality. The objective reduces to local policy matching, but estimating it still requires teacher rollouts in $M$.

### 3.2 Context Distillation with 1-Step Branch Rollout

Sampling enough teacher trajectories to distill a single $\tau^{\text{exp}}$ would itself require many additional rollouts in $M$. A natural alternative, inspired by model-based RL, is to replace $M$ with a learned world model $\widehat{M}$ and sample synthetic trajectories $\widetilde{\tau}' \sim p_{\widehat{M}}^{\theta_0}(\cdot \mid \tau^{\text{exp}})$ without further real interaction (Sutton & Barto, 2018; Chua et al., 2018). In text-only tasks, an LLM can serve as $\widehat{M}$, either through task-specific training or directly through in-context learning. However, world model errors compound over long rollouts (Talvitie, 2017; Janner et al., 2019), while predictions outside real-data support become unreliable (Yu et al., 2020), making long synthetic trajectories poor supervision.

Branched rollouts mitigate this problem by starting from real data and simulating only a short continuation (Janner et al., 2019; Lai et al., 2020; Hiraoka et al., 2021; Fan & Ramadge, 2021). Starting from a recorded branch point $h_t^{\text{exp}} = (o_0, a_0, \ldots, o_t)$, we sample the $k$-step continuation $\widetilde{\tau}_t^{(k)} \sim p_{\widehat{M}}^{\theta_0}(\cdot \mid h_t^{\text{exp}}, \tau^{\text{exp}})$, where $\widetilde{\tau}_t^{(k)} = (\widetilde{a}_t, \widetilde{o}_{t+1}, \widetilde{a}_{t+1}, \ldots, \widetilde{o}_{t+k-1}, \widetilde{a}_{t+k-1})$. Here $k$ counts teacher decisions, while $\tau^{\text{exp}}$ remains the separate experience context supplied to the teacher.

Smaller $k$ limits accumulated model error. We take the limiting case $k = 1$, where the branch contains only a teacher decision $a'_t \sim \pi_{\theta_0}(\cdot \mid h_t^{\text{exp}}, \tau^{\text{exp}})$, eliminating the world model. Since the teacher entropy is constant in $\theta$, optimizing the local forward KLs over recorded histories reduces to the following teacher-sampled objective.

$$\mathcal{L}_{\text{EPD}}(\theta; \tau^{\text{exp}}) = -\sum_{t=0}^{T-1} \mathrm{E}_{a'_t \sim \pi_{\theta_0}(\cdot \mid h_t^{\text{exp}}, \tau^{\text{exp}})}\left[\log \pi_\theta(a'_t \mid h_t^{\text{exp}})\right]. \tag{2}$$

Distilling $\tau^{\text{exp}}$ under this objective requires no additional environment interaction and therefore preserves environment sample efficiency.

### 3.3 Practical Implementation

Equation 2 defines the one-step distillation objective but leaves open how to turn long interaction histories into effective training data. Our implementation addresses this through experience preprocessing, enhanced teacher reasoning, and branch packing. Throughout this subsection, $a_t$ and $a'_t$ each denote a complete model output at one interaction turn, including both reasoning tokens and the final environment action.

**Experience Preprocessing.** Raw experience can be long and noisy: task-relevant evidence is often interleaved with repeated exploration, verbose environment responses, and details irrelevant to future decisions. This lowers the density of useful information available to the teacher, while sufficiently long histories may exceed its context window during target generation. We therefore preprocess the collected experience before teacher generation. We use $g(\tau^{\text{exp}})$ to denote the resulting teacher-facing context, where $g$ may rewrite, abstract, or summarize the original history to concentrate task-relevant information, preserve salient outcomes and feedback, and fit within the available context window.

**Enhanced Teacher Reasoning.** We empirically find that eliciting longer reasoning from the teacher improves post-distillation performance, as shown in Appendix A.1. Motivated by this observation, we include a fixed reasoning prompt $I$ during teacher generation. The prompt instructs the teacher to examine the preprocessed experience $g(\tau^{\text{exp}})$ and reason more thoroughly before producing its next decision.

**Branch Packing.** A direct implementation of Equation 2 creates a separate training example $(h_t^{\text{exp}}, a'_t)$ at each branch point. This has low supervision density because long, overlapping history prefixes are repeatedly processed even though only the tokens in $a'_t$ contribute to the loss. We instead pack branches from successive points of the same recorded trajectory into a single training sequence. For $\tau^{\text{exp}} = (o_0, a_0, \ldots, a_{T-1}, o_T)$, the packed sequence is

$$c_T = (o_0, a'_0, a_0, o_1, \ldots, a'_{T-1}, a_{T-1}, o_T),$$

where $c_t$ denotes its prefix ending at $o_t$. For simplicity, we omit from the notation the prompt annotations used in practice to distinguish recorded actions $a_t$ from teacher-generated decisions $a'_t$.

Starting from $c_0 = (o_0)$, the teacher samples $a'_t \sim \pi_{\theta_0}(\cdot \mid c_t, g(\tau^{\text{exp}}), I)$ at each branch point and then appends the recorded pair $(a_t, o_{t+1})$. Because this pair is copied from $\tau^{\text{exp}}$, $o_{t+1}$ is the response to $a_t$, not $a'_t$. Repeating this process induces

$$q_{\theta_0}(c_T \mid \tau^{\text{exp}}) = \prod_{t=0}^{T-1} \pi_{\theta_0}(a'_t \mid c_t, g(\tau^{\text{exp}}), I). \tag{3}$$

Because $c_t$ contains earlier teacher decisions, branch packing is not an exact implementation of the objective in Equation 2, but a practical approximation. Under the packed-sequence distribution in Equation 3, the student objective becomes

$$\mathcal{L}_{\text{EPD}}(\theta; \tau^{\text{exp}}) = -\mathrm{E}_{c_T \sim q_{\theta_0}(\cdot \mid \tau^{\text{exp}})} \left[ \sum_{t=0}^{T-1} \log \pi_\theta(a'_t \mid c_t) \right]. \tag{4}$$

Only the teacher-generated decisions $a'_{0:T-1}$ contribute to the loss, while the recorded observations and actions serve as context. Experiments reported in Table 3 show that this approximation retains distillation performance while substantially reducing teacher-generation and training time.

## 4 Experiments

Our experiments first evaluate how effectively Experience Distillation distills the ICL gains induced by task-specific experience into model weights (Section 4.2). We next assess the environment-sample efficiency of in-context learning from trial-and-error experience followed by Experience Distillation (Section 4.3). Subsequent experiments evaluate the model-free one-step formulation and the efficiency of branch packing (Sections 4.4 and 4.5). We then test whether capabilities distilled from multi-task experience generalize to OOD tasks (Section 4.6). We also examine whether Experience Distillation can operate in a continual setting (Section 4.7). Finally, we compare teacher-sampled forward KL with on-policy distillation (Section 4.8), evaluate sampled NTP against direct logit KL (Section 4.9), and characterize the scaling and cost of teacher-generated data (Section 4.10).

### 4.1 Experimental Setting

**Tasks.** We evaluate on two domains. TaleSuiteJericho (Cui et al., 2025) provides long-horizon text-adventure tasks in which an agent acts through natural-language commands and receives environment observations and game scores. We also construct 749 curated software-engineering (SWE) tasks, where an agent receives a task issue and repository and can inspect code, edit files, run commands and tests, and observe commit feedback. TaleSuite experiments use fixed subsets of a seven-task pool: Acorncourt, Balances, Detective, Enter, Inhumane, Library, and Reverb.

**Experience.** On both domains, we construct task-specific experience by allowing the agent to attempt the same task multiple times and condition each later trial on the preceding trials. In TaleSuite, the game state resets between trials. In curated SWE, successive trials preserve the interaction history and evolving working tree, allowing the agent to revise its repair after tool, test, and commit feedback. The resulting multi-trial record forms the experience context for that task. Appendix A.7 reports the TaleSuite task subsets, collection protocols, and experience-length statistics.

**Evaluation.** For each method in both domains, we report two metrics: domain-specific task performance and the ICL-normalized gain $G_{\text{ICL}}$. Task performance is average pass@1 over all 749 curated SWE tasks and average normalized score over the evaluated TaleSuite tasks; Appendix A.7 provides details on TaleSuite score normalization. For the ICL reference, the base model conditions

on the collected experience for the task being evaluated, without updating its weights. This measures the performance gain available when the task-specific experience remains in context. To quantify how much of this gain is achieved by methods that update model weights using the same experience, we define

$$G_{\mathrm{ICL}}(m) = \frac{S_m - S_{\mathrm{ZS}}}{S_{\mathrm{ICL}} - S_{\mathrm{ZS}}} \times 100\%.$$

Here, $S_m$ is the performance of method $m$, $S_{\mathrm{ZS}}$ is zeroshot performance, and $S_{\mathrm{ICL}}$ is the ICL reference, using the corresponding domain metric defined above. This normalization assigns zeroshot 0% and ICL 100%. Table 1 computes $G_{\mathrm{ICL}}$ from the reported cross-task average scores. Task-level ablations instead compute $G_{\mathrm{ICL}}$ separately for each task and report the mean task-level gain.

**Training and Evaluation Protocol.** We compare Experience Distillation with ICL, SFT, and classical RL methods. ICL, SFT, and Experience Distillation use the same collected experience. At evaluation, only ICL conditions on the experience for the corresponding task; all methods that update model weights are evaluated without the experience context. SFT trains directly on the collected trajectories; for Experience Distillation, the teacher and student start from the same checkpoint, and only the teacher receives the experience context during target generation. We use distinct in-house base models for curated SWE and TaleSuite. Unless noted otherwise, Experience Distillation uses multi-task joint training, and all curated SWE experiments use the branch-packed training sequences described in Section 3.3. In the sample-efficiency comparison, one environment sample denotes one complete agent trial or rollout rather than one interaction step. For curated SWE, the environment-sample cost reported for ICL includes the full exploration cost incurred to identify the retained successful trajectory. Curated SWE results are averaged over 10 runs; unless otherwise stated, TaleSuite results are averaged over more than 16 runs.

### 4.2 Distilling Multi-Task Experience into Model Weights

**Objective.** We first test whether Experience Distillation can consolidate the performance gain induced by task-specific experience in context into model weights, and how much of that gain it retains. Table 1 compares the three methods using the domain-specific performance metrics and $G_{\mathrm{ICL}}$ defined in Section 4.

| Method | 749 curated SWE tasks | | TaleSuite (6 tasks) | |
|---|---|---|---|---|
| | Avg. pass@1 | $G_{\mathrm{ICL}}$ | Avg. normalized score | $G_{\mathrm{ICL}}$ |
| ICL (reference) | 76.4% | 100.0% | 45.6 | 100.0% |
| Zeroshot | 5.3% | 0.0% | 18.5 | 0.0% |
| SFT | 8.0% | 3.8% | 17.8 | −2.6% |
| Experience Distillation | **51.4%** | **64.8%** | **43.8** | **93.4%** |

Table 1: Experience Distillation on 749 curated SWE tasks and six TaleSuite tasks. Training uses branch-packed sequences and multi-task joint training; results are averaged over 10 and 16 runs, respectively.

**Main finding.** On curated SWE, Experience Distillation reaches 51.4% average pass@1 and retains 64.8% of the ICL gain. On TaleSuite, it reaches an average normalized score of 43.8 and retains 93.4% of the ICL gain. In contrast, SFT achieves $G_{\mathrm{ICL}} = 3.8\%$ and $-2.6\%$ on curated SWE and TaleSuite, respectively. These results show that Experience Distillation can consolidate a substantial fraction of experience-induced ICL gains into model weights, whereas directly training on the collected trajectories yields negligible improvement in model performance.

**Additional results.** Appendix A.4 shows the same trend under pass@10, while Appendices B.3 and C illustrate experience-derived behavior exhibited by Experience Distillation-trained models without access to the experience context at evaluation.

### 4.3 Environment-Sample Efficiency of ICL Followed by Experience Distillation

**Objective.** We compare the environment-sample efficiency of in-context learning from trial-and-error experience followed by Experience Distillation against classical RL methods. We use

PPO (Schulman et al., 2017) on curated SWE, motivated by its recent use in long-horizon coding-agent training (Z.ai, 2026), and GRPO (Shao et al., 2024) on TaleSuite. For both ICL + Experience Distillation and ICL + SFT, all environment-sample cost comes from the ICL experience-collection stage; Experience Distillation and SFT introduce no additional environment sampling. For each RL baseline, we report the best observed performance within the complete training run and charge that result to the run's full environment-sample budget. Figure 1 plots these budgets against the corresponding domain performance.

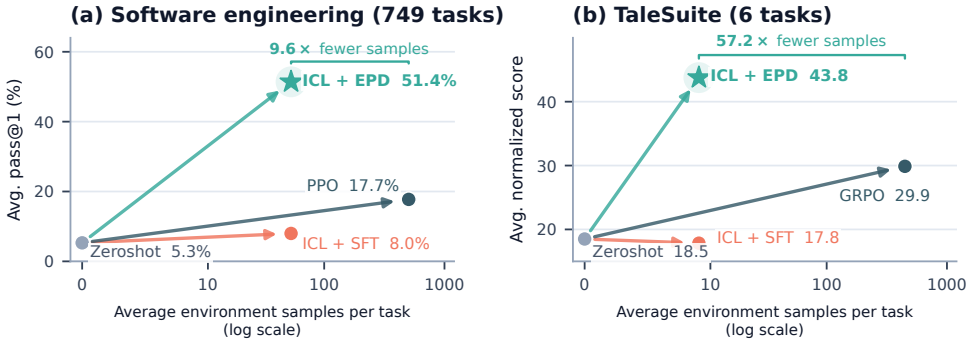


Figure 1: Environment-sample efficiency of ICL followed by Experience Distillation (ICL + EPD) and RL baselines. Left: average pass@1 on 749 curated SWE tasks; right: average normalized score on six TaleSuite tasks, both versus average environment samples per task. RL points pair the best observed performance with the complete run budget. Arrows show gains from zeroshot, and brackets compare the ICL + EPD and RL sample budgets. The x-axis uses a $\log(1 + x)$ transform, with tick labels in the original sample counts.

**Main finding.** On curated SWE, ICL followed by Experience Distillation reaches 51.4% average pass@1, compared with 17.7% for PPO, while using $9.6\times$ fewer environment samples. On TaleSuite, the same learning pipeline reaches an average normalized score of 43.8, compared with 29.9 for GRPO, while using $57.2\times$ fewer samples. Thus, combining ICL with Experience Distillation achieves approximately an order-of-magnitude reduction in environment samples relative to the RL baselines.

**Additional results.** Appendices B.1 and B.4 report the full GRPO and PPO learning curves, respectively, while Appendix B.3 provides qualitative comparisons on TaleSuite that reveal distinct model-output patterns after GRPO and Experience Distillation training.

### 4.4 Evaluating Model-Free One-Step Distillation

**Objective.** We evaluate the effect of the model-free one-step formulation by comparing it with model-based implementations. Table 2 compares three rollout strategies with progressively less world-model participation. In the model-based full rollout, the teacher generates up to 100 decisions, with a world-model observation between successive decisions. The model-based branch rollout contains two teacher decisions separated by one world-model observation. The model-free one-step rollout generates only one teacher decision and no future observation.

| | | | **Model-free** | **Model-based** | |
|---|---|---|---|---|---|
| **Task** | **Zeroshot** | **ICL** | **1-step branch rollout** | **Branch rollout** | **Full rollout** |
| Acorncourt | 17.2 | 38.5 | 34.9 / 83.1% | 53.6 / 170.9% | – |
| Balances | 15.9 | 39.6 | 34.9 / 80.2% | 21.6 / 24.1% | – |
| Detective | 31.3 | 90.8 | 82.9 / 86.7% | 71.8 / 68.1% | 42.7 / 19.2% |
| Enter | 40.2 | 68.5 | 59.7 / 68.9% | 38.5 / -6.0% | – |
| Inhumane | 5.9 | 36.5 | 36.5 / 100.0% | 14.9 / 29.4% | – |
| **Avg.** | **22.1** | **54.8** | **49.8 / 83.8%** | **40.1 / 57.3%** | – |

Table 2: Model-free and model-based Experience Distillation variants using separate branch examples and multi-task joint training on five TaleSuite tasks. Method cells report normalized score / task-level $G_{\mathrm{ICL}}$; the average row reports their means.

**Main finding.** Among the evaluated variants, model-free one-step distillation performs best on average. Across five tasks, it reaches an average task-level $G_{\mathrm{ICL}}$ of 83.8%, compared with 57.3% for model-based branch rollout. On Detective, shortening full rollout to branch rollout raises $G_{\mathrm{ICL}}$ from 19.2% to 68.1%, while removing the world-model observation raises it to 86.7%. In this comparison, reducing world-model participation and rollout length improves the average effectiveness of the distillation targets.

**Additional results.** Appendix A.5 provides qualitative examples of errors in model-based rollouts.

### 4.5 Efficient Branch Packing for Language Agents

**Objective.** We evaluate whether the practical branch-packing approximation improves supervision density and overall training efficiency relative to a direct implementation of the one-step distillation objective. The direct implementation creates a separate training example at each recorded branch point. Branch packing instead generates teacher decisions autoregressively across successive branch points and places them in a single training sequence, as described in Section 3.3. Table 3 compares these two implementations.

| | **Separate branch examples** | **Branch-packed sequences** |
|---|---|---|
| Generated training instances | 4096 examples | 128 sequences |
| Training steps | 768 | 64 |
| Total time (normalized) | $> 10.0$ | 1.0 |
| **Task** | **Normalized score / $G_{\mathrm{ICL}}$** | |
| Balances | 37.8 / 92.6% | 38.8 / 96.6% |
| Detective | 84.3 / 89.1% | 88.8 / 96.6% |
| Enter | 59.9 / 69.7% | 61.5 / 75.3% |
| Inhumane | 41.5 / 116.3% | 36.6 / 100.3% |
| Library | 33.3 / 100.2% | 33.3 / 100.0% |
| Reverb | 1.9 / 37.6% | 3.6 / 72.0% |
| Avg. | 43.1 / 84.2% | 43.8 / 90.1% |

Table 3: Branch-packing ablation for Experience Distillation on six TaleSuite tasks. Total time includes teacher generation and student training and is normalized to branch packing. Task cells report normalized score / task-level $G_{\mathrm{ICL}}$, averaged over 16 runs.

**Main finding.** Branch packing replaces 4096 separate branch examples with 128 packed sequences, each supervising multiple teacher decisions. It reduces the number of training steps from 768 to 64 and the combined teacher-generation and student-training time by more than an order of magnitude. Performance remains comparable across the six tasks: the average normalized score changes from 43.1 to 43.8, and the average task-level $G_{\mathrm{ICL}}$ from 84.2% to 90.1%.

### 4.6 Generalization to OOD Tasks

**Objective.** We evaluate whether the capability distilled from multi-task experience transfers to new tasks not represented in the collected experience. We evaluate the model trained on the 749 curated SWE tasks on 494 OOD software-engineering tasks excluded from both experience collection and distillation. These tasks comprise 278 cross-repository OOD tasks whose repositories and task types are both unseen and 216 within-repository OOD tasks that use code repositories or software libraries represented in the collected experience but present new task types. We report pass@1 and pass@5.

| Subset | Method | pass@1 | pass@5 |
|---|---|---|---|
| Full OOD set (494) | Zeroshot | 4.62% | 20.39% |
| | EPD | **8.84%** | **26.13%** |
| Cross-repository OOD (278) | Zeroshot | 4.72% | 21.40% |
| | EPD | **8.87%** | **27.22%** |
| Within-repository OOD (216) | Zeroshot | 4.49% | 19.08% |
| | EPD | **8.80%** | **24.73%** |

Table 4: Generalization of multi-task Experience Distillation (EPD) to cross-repository and within-repository OOD software-engineering tasks.

**Main finding.** On the full 494-task OOD set, Experience Distillation improves pass@1 from 4.62% to 8.84% and pass@5 from 20.39% to 26.13%. The improvement appears at both levels of distribution shift, including cross-repository OOD tasks whose repositories and task types are unseen during experience collection. These results show that the capability distilled from multi-task experience transfers beyond the tasks represented in that experience.

### 4.7 Continual Experience Distillation

**Objective.** We evaluate whether performance improvements accumulate across repeated Experience Distillation cycles. The preceding experiments use one collect-and-distill cycle; *Continual Experience Distillation* repeats this process, with the checkpoint produced by each cycle initializing the next cycle while the previous experience context is cleared. Each cycle collects four repeated trials separately for each task and performs one multi-task distillation update.

Figure 2 reports results on six TaleSuite tasks. Cycle 0 denotes the initial checkpoint before any collect-and-distill cycle.

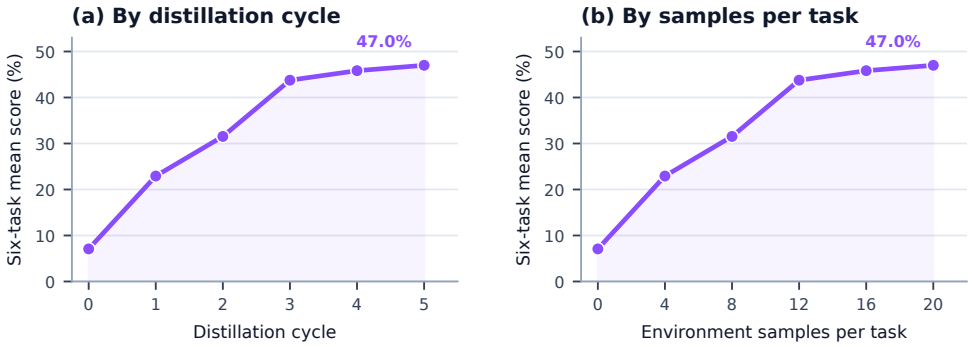


Figure 2: Continual Experience Distillation on six TaleSuite tasks. Cycle 0 is the initial checkpoint; each subsequent cycle collects four repeated trials per task and performs one distillation update.

**Main finding.** The mean normalized score across the six tasks increases from 7.1 at cycle 0 to 47.0 after five cycles, corresponding to 20 cumulative environment samples per task. Because each cycle discards the preceding experience context and continues from the updated checkpoint, this improvement shows that experience-derived gains can accumulate across cycles through model weights.

### 4.8 Teacher-Sampled Forward KL versus Student-Sampled Reverse KL

**Objective.** By default, our implementation of Experience Distillation uses teacher-sampled forward KL. The experience-conditioned teacher generates branch-packed training sequences, which train the student without the experience context. We investigate an alternative based on student-sampled reverse KL, following prior work on on-policy distillation (Agarwal et al., 2024; Lu & Thinking Machines Lab, 2025; Ye et al., 2026b; Penaloza et al., 2026). Under OPD, the student generates trajectories without the experience context, and the teacher provides per-token reverse-KL supervision on those trajectories. We compare these two procedures on Balances and Detective using single-task models and matched compute.

| Task | Zeroshot | ICL | Forward KL (Default) | Reverse KL (OPD) |
|---|---|---|---|---|
| Balances | 0.0% | 100.0% | **96.7%** | 9.1% |
| Detective | 0.0% | 100.0% | **98.4%** | 0.4% |

Table 5: The default teacher-sampled forward-KL implementation of Experience Distillation versus a student-sampled reverse-KL variant based on on-policy distillation (OPD). Both use single-task training and matched compute on two TaleSuite tasks. Values report $G_{\mathrm{ICL}}$, averaged over 16 runs.

**Main finding.** The default teacher-sampled forward-KL implementation nearly matches ICL, reaching $G_{\mathrm{ICL}}$ of 96.7% on Balances and 98.4% on Detective. In contrast, the student-sampled reverse-KL variant remains near zeroshot at 9.1% and 0.4%. We hypothesize that, without the experience context, the student rarely samples trajectories containing the high-quality decisions induced by that context. Reverse-KL supervision along those student trajectories then does not directly include the missing decisions, limiting their transfer. Appendix A.6 provides qualitative examples after OPD training.

### 4.9 Sampled-Token versus Full-Distribution Distillation

**Objective.** Our default implementation samples an action from the teacher and trains the student with next-token prediction by minimizing cross-entropy on the sampled tokens. This retains only one target token at each position. An alternative is to record the teacher's full next-token distribution at every position along the sampled action and directly match the student's corresponding distribution to it. This provides denser supervision over the vocabulary but incurs substantially greater memory overhead because the full teacher logits must be stored and processed. Using the same one-step branch construction, we compare the performance of these two training objectives.

| Task | Zeroshot | ICL | Sampled-Token NTP (Default) | Full-Distribution KL |
|---|---|---|---|---|
| Balances | 15.9 | 39.6 | 37.8 / 92.6% | 37.7 / 91.9% |
| Detective | 31.3 | 90.8 | 84.3 / 89.1% | 92.8 / 103.3% |
| Enter | 40.2 | 68.5 | 59.9 / 69.7% | 58.4 / 64.2% |
| Inhumane | 5.9 | 36.5 | 41.5 / 116.3% | 37.7 / 103.8% |
| Library | 17.7 | 33.3 | 33.3 / 100.2% | 32.0 / 91.9% |
| Reverb | 0.0 | 5.0 | 1.9 / 37.6% | 1.8 / 36.8% |
| **Avg.** | **18.5** | **45.6** | **43.1 / 84.2%** | **43.4 / 82.0%** |

Table 6: Sampled-token NTP versus full-distribution KL for Experience Distillation, using separate one-step branch examples. Method cells report normalized score / task-level $G_{\mathrm{ICL}}$; results are averaged over 16 runs after 16 training epochs.

**Main finding.** Full-distribution KL provides no clear performance gain over sampled-token NTP. The two variants achieve similar average normalized scores, while sampled-token NTP obtains a slightly higher mean task-level $G_{\mathrm{ICL}}$ (84.2% versus 82.0%) and avoids storing teacher logits for long agent contexts. We therefore use sampled-token NTP as the scalable default in the main experiments.

### 4.10 Scaling and Cost of Teacher-Generated Data

**Objective.** Holding the collected experience fixed, we scale the number of branch-packed sequences generated per experience trial from 1 to 16 to test whether additional teacher-generated supervision improves distillation without further environment interaction.

**Main finding.** Figure 3 shows that this increase, corresponding to teacher-generated data totaling $0.17\times$ to $2.79\times$ the 61.7M-token experience corpus, raises $G_{\mathrm{ICL}}$ from 31.8% to 64.8%. The current teacher-generation cost remains a limitation of our method, and its efficiency may be further improved.

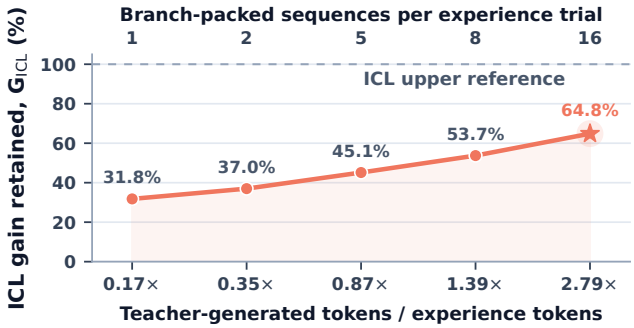


Figure 3: Teacher-generated data scaling for Experience Distillation on 749 curated SWE tasks. The upper x-axis reports branch-packed sequences per experience trial; the lower reports teacher-generated tokens relative to the 61.7M-token experience corpus.

# 5 Related Work

## 5.1 Context Distillation

Context distillation builds on knowledge distillation (Hinton et al., 2015) and transfers behavior induced by an informative context into model parameters, so a student can reproduce a context-conditioned teacher without repeatedly consuming that context. Early alignment work used this idea for prompt-conditioned behavior (Askell et al., 2021), and Snell et al. (2022) formalized it for instructions, demonstrations, and scratchpads. Related work distills few-shot in-context learning ability, rationales, and long contexts into smaller models, adapters, or latent memories (Huang et al., 2022; Hsieh et al., 2023; Duan et al., 2024; Charakorn et al., 2026; Zheng et al., 2026). This view is also related to learning with privileged information, where the teacher observes information available during training but absent at deployment (Lopez-Paz et al., 2016). Recent work moves closer to agents: Ye et al. (2026b) propose on-policy context distillation for experiential knowledge and system prompts, and Penaloza et al. (2026) study privileged-information distillation in multi-turn agentic environments. Experience Distillation addresses a distinct agent-learning problem: the context is a real agent-environment trajectory, and the desired target is an improved future trajectory distribution. Directly sampling that future trajectory costs additional environment interaction, while simulating it with a world model introduces rollout error. We therefore use one-step branch rollouts from real histories, turning scarce interaction experience into supervised decisions without future environment rollouts or long world-model rollouts.

## 5.2 In-Context Learning

In-context learning enables a fixed-parameter model to adapt from demonstrations and other task-specific evidence supplied at inference time (Brown et al., 2020; Min et al., 2022; Garg et al., 2022). Prior work has studied how this adaptation arises through implicit Bayesian inference, induction heads, and learning algorithms implemented in transformer activations (Xie et al., 2022; Olsson et al., 2022; Akyürek et al., 2023; Von Oswald et al., 2023). In sequential decision making, recurrent meta-RL uses observation-action-feedback histories for fast adaptation (Duan et al., 2016; Wang et al., 2016). Transformer-based methods model cross-episode histories to perform task inference, exploration, and policy improvement in context (Melo, 2022; Laskin et al., 2023; Lee et al., 2023; Grigsby et al., 2024), while Nikulin et al. (2025) scale this setting to large collections of learning histories. Language agents similarly reuse self-collected experience through reflections, extracted insights, semantic memories, and reusable workflows (Shinn et al., 2023; Zhao et al., 2024; Majumder et al., 2024; Gupta et al., 2024; Wang et al., 2025). Although these approaches represent experience differently, their adaptation remains mediated by inference-time context or external memory rather than persistent parameter updates. Experience Distillation studies the complementary problem of consolidating the behavioral gains elicited by collected experience into model weights, so the experience need not remain available at inference time.

### 5.3 Sample-Efficient Reinforcement Learning

Achieving strong performance with fewer environment interactions has long been a central challenge in reinforcement learning. Existing approaches reuse collected data through off-policy learning and experience replay (Haarnoja et al., 2018; Andrychowicz et al., 2017), generate synthetic rollouts with learned world models (Chua et al., 2018; Janner et al., 2019), or learn from fixed offline datasets (Levine et al., 2020; Kumar et al., 2020; Kostrikov et al., 2022; Chen et al., 2021). These ideas have also been extended to language agents through off-policy multi-turn RL, synthetic environments, and richer supervision extracted from interaction feedback (Zhou et al., 2024; Chen et al., 2026; Zhang et al., 2026; Hübotter et al., 2026). These methods improve a policy through reward or value estimation, additional observed transitions, or learned environment dynamics. Our implementation of Experience Distillation instead uses the model's in-context learning over self-collected experience as the source of improved decisions, then distills those decisions at recorded history points without estimating returns, predicting future observations, or collecting additional environment interactions.

## 6 Conclusion

This work studies Experience Distillation, the problem of consolidating what agents learn from interaction histories into model weights while preserving environment sample efficiency. By re-sampling only the teacher's next decision at histories already observed in the collected experience, our implementation requires neither further environment interaction nor a world model. Across software-engineering tasks and text-adventure games, it retains substantial gains from ICL, clearly outperforms direct SFT, and matches classical RL baselines with at least $9.6\times$ fewer environment samples. Results on OOD tasks and repeated collect-and-distill cycles further indicate that distilled capabilities can transfer to new tasks and that experience-derived gains can accumulate across cycles. More broadly, these findings suggest a learning paradigm in which agents first learn quickly from a small amount of trial-and-error experience in context and then periodically consolidate what they have learned into model weights.

## REFERENCES


Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, 2024.

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models. In *International Conference on Learning Representations*, 2023.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Jackson Kernion, Kamal Ndousse, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, and Jared Kaplan. A general language assistant as a laboratory for alignment. *arXiv preprint arXiv:2112.00861*, 2021.

Tom B. Brown et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020.

Rujikorn Charakorn, Edoardo Cetin, Shinnosuke Uesaka, and Robert Tjarko Lange. Doc-to-LoRA: Learning to instantly internalize contexts. *arXiv preprint arXiv:2602.15902*, 2026.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Zhaorun Chen et al. Scaling agent learning via experience synthesis. In *International Conference on Learning Representations*, 2026.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Christopher Zhang Cui, Xingdi Yuan, Ziang Xiao, Prithviraj Ammanabrolu, and Marc-Alexandre Côté. TALES: Text adventure learning environment suite. *arXiv preprint arXiv:2504.14128*, 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.

Yifei Duan, Liu Li, Zirui Zhai, and Jinxia Yao. In-context learning distillation for efficient few-shot fine-tuning. *arXiv preprint arXiv:2412.13243*, 2024.

Ting-Han Fan and Peter J. Ramadge. A contraction approach to model-based reinforcement learning. In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pp. 325–333, 2021.

Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. In *Advances in Neural Information Processing Systems*, volume 35, pp. 30583–30598, 2022.

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

Daya Guo et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645:633–638, 2025. doi: 10.1038/s41586-025-09422-z.

Priyanshu Gupta, Shashank Kirtania, Ananya Singha, Sumit Gulwani, Arjun Radhakrishna, Gustavo Soares, and Sherry Shi. MetaReflection: Learning instructions for language agents using past reflections. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8369–8385, 2024.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 1861–1870, 2018.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Takuya Hiraoka, Takahisa Imagawa, Voot Tangkaratt, Takayuki Osa, Takashi Onishi, and Yoshimasa Tsuruoka. Meta-model-based meta-policy optimization. In *Proceedings of the 13th Asian Conference on Machine Learning*, volume 157 of *Proceedings of Machine Learning Research*, pp. 129–144, 2021.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. *arXiv preprint arXiv:2305.02301*, 2023.

Yukun Huang, Yanda Chen, Zhou Yu, and Kathleen McKeown. In-context learning distillation: Transferring few-shot learning ability of pre-trained language models. *arXiv preprint arXiv:2212.10670*, 2022.

Jonas Hübotter et al. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-based reinforcement learning for Atari. In *International Conference on Learning Representations*, 2020.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit Q-learning. In *International Conference on Learning Representations*, 2022.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian Goodrich, Nikola Jurkovic, Luke Harold Miles, Seraphina Nix, Tao Lin, Chris Painter, Neev Parikh, David Rein, Lucas Jun Koba Sato, Hjalmar Wijk, Daniel M. Ziegler, Elizabeth Barnes, and Lawrence Chan. Measuring AI ability to complete long software tasks. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Hang Lai, Jian Shen, Weinan Zhang, and Yong Yu. Bidirectional model-based policy optimization. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5618–5627, 2020.

Michael Laskin et al. In-context reinforcement learning with algorithm distillation. In *International Conference on Learning Representations*, 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*, volume 36, 2023.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

David Lopez-Paz, Léon Bottou, Bernhard Schölkopf, and Vladimir Vapnik. Unifying distillation and privileged information. In *International Conference on Learning Representations*, 2016.

Kevin Lu and Thinking Machines Lab. On-policy distillation. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20251026. https://thinkingmachines.ai/blog/on-policy-distillation.

Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. CLIN: A continually learning language agent for rapid task adaptation and generalization. In *Conference on Language Modeling*, 2024.

Luckeciano C. Melo. Transformers are meta-reinforcement learners. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 15340–15359, 2022.

METR. Task-completion time horizons of frontier AI models. https://metr.org/time-horizons/, May 8 2026.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11048–11064, 2022.

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21(181):1–50, 2020.

Alexander Nikulin, Ilya Zisman, Alexey Zemtsov, and Vladislav Kurenkov. XLand-100B: A large-scale multi-task dataset for in-context reinforcement learning. In *International Conference on Learning Representations*, 2025.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

Emiliano Penaloza, Dheeraj Vattikonda, Nicolas Gontier, Alexandre Lacoste, Laurent Charlin, and Massimo Caccia. Privileged information distillation for language models. *arXiv preprint arXiv:2602.04942*, 2026.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

David Silver et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489, 2016. doi: 10.1038/nature16961.

Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context. *arXiv preprint arXiv:2209.15189*, 2022.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

Erik Talvitie. Self-correcting models for model-based reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Joel Veness, Kee Siong Ng, Marcus Hutter, and David Silver. Reinforcement learning via AIXI approximation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pp. 605–611, 2010.

Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, Joao Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 35151–35174, 2023.

Hao Wang, Guozhi Wang, Han Xiao, Yufeng Zhou, Yue Pan, Jichao Wang, Ke Xu, Yafei Wen, Xiaohu Ruan, Xiaoxin Chen, and Honggang Qi. Skill-SD: Skill-Conditioned Self-Distillation for Multi-turn LLM Agents. *arXiv preprint arXiv:2604.10674*, 2026.

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

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 63897–63911, 2025.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. In *International Conference on Learning Representations*, 2022.

Tianzhu Ye, Li Dong, Qingxiu Dong, Xun Wu, Shaohan Huang, and Furu Wei. Online experiential learning for language models. *arXiv preprint arXiv:2603.16856*, 2026a.

Tianzhu Ye, Li Dong, Xun Wu, Shaohan Huang, and Furu Wei. On-policy context distillation for language models. *arXiv preprint arXiv:2602.12275*, 2026b.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y. Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. MOPO: Model-based offline policy optimization. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Z.ai. GLM-5.2: Built for long-horizon tasks. https://z.ai/blog/glm-5.2, June 16 2026.

Kai Zhang et al. Agent learning via early experience. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, 2026.

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

Ziyang Zheng, Zeju Li, Xiangyu Wen, Jianyuan Zhong, Junhua Huang, Lei Chen, Mingxuan Yuan, and Qiang Xu. Context distillation as latent memory management. *arXiv preprint arXiv:2605.28889*, 2026.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. ArCHer: Training language model agents via hierarchical multi-turn RL. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, 2024.

Deyao Zhu et al. EdgeBench: Unveiling scaling laws of learning from real-world environments. *arXiv preprint arXiv:2607.05155*, 2026.

# Appendix

## Appendix Overview

Appendix A provides additional Experience Distillation ablations, TaleSuite frontier-model references, curated SWE pass@10 results, qualitative diagnostics of model-based and on-policy distillation, and detailed experience-collection statistics. Appendix B analyzes the comparison between ICL followed by Experience Distillation and RL baselines through aggregate and task-level GRPO learning curves, a controlled ColorButton exploration diagnostic, matched TaleSuite behavior comparisons, and the complete PPO training trajectory. Appendix C presents detailed SWE case studies tracing task-specific knowledge from accumulated experience to teacher-generated decisions and to Experience Distillation-trained model behavior without the experience context at evaluation.

## A Additional Experimental Details and Results

### A.1 Effect of Enhanced Teacher Reasoning

Section 3.3 introduces Enhanced Teacher Reasoning to elicit more thorough teacher decisions from the preprocessed experience context. We evaluate its effect on Detective by disabling long teacher reasoning while holding the teacher-generation budget fixed. Table 7 reports the normalized game score and $G_{\mathrm{ICL}}$, which maps zeroshot to 0% and ICL to 100%.

| Metric | Zeroshot | EPD w/ enhanced reasoning | EPD w/o enhanced reasoning | ICL |
|---|---|---|---|---|
| Normalized score | 31.1 | **71.8** | 50.7 | 87.2 |
| $G_{\mathrm{ICL}}$ | 0.0% | **72.5%** | 34.9% | 100.0% |

Table 7: Effect of Enhanced Teacher Reasoning during Experience Distillation (EPD) target generation on Detective. Both variants use 4096 teacher rollouts, three rollout steps, and one to five collected histories in context.

**Finding.** Enhanced Teacher Reasoning raises the normalized score from 50.7 to 71.8 and $G_{\mathrm{ICL}}$ from 34.9% to 72.5%. The result supports eliciting more thorough teacher reasoning when generating distillation targets from the experience context.

### A.2 Effect of Learning Rate

We examine how the optimization schedule affects the transfer from teacher-generated training data to final task performance. Figure 4 compares two learning rates across checkpoints with different numbers of training epochs. The horizontal axis reports training loss and is reversed, so lower loss appears farther to the right; the vertical axis reports $G_{\mathrm{ICL}}$ on a 0–1 scale.

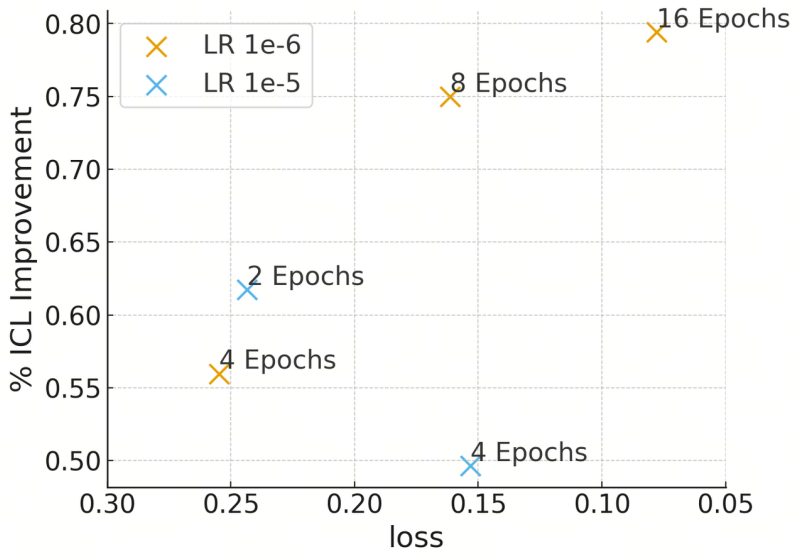


Figure 4: Effect of learning rate and training duration on Experience Distillation (EPD). Crosses mark checkpoints, and adjacent labels give the number of training epochs. $G_{\mathrm{ICL}}$ is plotted as a fraction, so 0.80 corresponds to 80%.

**Finding.** The smaller learning rate with more epochs yields the strongest evaluation performance among the evaluated schedules. At similar training losses around 0.15–0.16, the $10^{-6}$ run after 8 epochs substantially outperforms the $10^{-5}$ run after 4 epochs. Further fitting at $10^{-5}$ lowers training loss but degrades evaluation performance, whereas the $10^{-6}$ checkpoints improve from 4 to 16 epochs as training loss decreases. Final training loss alone therefore does not determine distillation quality.

### A.3 TALESUITE FRONTIER-MODEL REFERENCE RESULTS

Table 8 places the improvement from ICL followed by Experience Distillation alongside inference-only zeroshot results from several frontier models under the same six-task TaleSuite evaluation protocol.

| System / setting | Balances | Detective | Enter | Inhumane | Library | Reverb | Avg. |
|---|---|---|---|---|---|---|---|
| *Inference-only zeroshot references* | | | | | | | |
| Claude Opus 4.5 | 29.41 | 33.80 | 79.17 | 79.63 | 40.56 | 0.00 | 43.76 |
| GPT-5.2 | 10.13 | 11.11 | 62.50 | 44.44 | 33.33 | 0.00 | 26.92 |
| Gemini 2.5 Pro | 14.71 | 30.09 | 66.67 | 22.22 | 33.33 | 0.00 | 27.84 |
| Gemini 3 Pro | 13.07 | 33.80 | 79.17 | 27.78 | 34.44 | 0.00 | 31.38 |
| Kimi-K2-Thinking | 16.34 | 30.09 | 75.00 | 7.41 | 22.22 | 0.00 | 25.18 |
| *Our model* | | | | | | | |
| Zeroshot | 15.90 | 31.30 | 40.20 | 5.90 | 17.70 | 0.00 | 18.50 |
| ICL + EPD | 38.80 | 88.80 | 61.50 | 36.60 | 33.30 | 3.60 | 43.77 |

Table 8: Frontier-model reference results on six TaleSuite tasks. Frontier-model rows are inference-only zeroshot evaluations. The ICL + EPD row reports the trained model evaluated without the experience context. Scores use the normalization defined in Appendix A.7.

**Finding.** ICL followed by Experience Distillation raises the average normalized score of our model from 18.50 to 43.77. Under the reported six-task protocol, this result is comparable to Claude Opus 4.5 at 43.76 and exceeds the other frontier-model references in Table 8. Experience Distillation therefore elevates the base model to the performance range of frontier references on these tasks.

### A.4 CURATED SWE PASS@10 RESULTS

Table 9 complements the pass@1 comparison in Table 1 with pass@10 over ten independently sampled candidates per task. Only the ICL reference receives the task-specific experience context at evaluation.

| Method | Avg. pass@1 | pass@10 |
|---|---|---|
| ICL (upper reference) | 76.4% | 86.4% |
| Zeroshot | 5.3% | 31.9% |
| SFT | 8.0% | 39.8% |
| EPD | **51.4%** | **80.9%** |

Table 9: Pass@1 and pass@10 on 749 curated SWE tasks. Pass@10 measures whether at least one of ten independently sampled candidates per task is accepted. Only ICL receives the task-specific experience context at evaluation.

### A.5 Observed Errors in Model-Based Rollouts

To examine qualitative errors associated with the model-based results in Table 2, we inspect representative TaleSuite rollouts generated by the world-model role. The action-generation and observation-modeling roles use the same underlying model weights with different prompts. Both excerpts come from generated Detective rollouts around the closet state. The first follows a teacher decision with a world-model observation and a subsequent teacher decision; it shows a fabricated reward changing the next action. The second shows policy-output tokens embedded in a generated observation, breaking the separation between environment observations and policy decisions. These excerpts diagnose rollout errors rather than provide a complete Experience Distillation evidence chain.

**Model-Based Detective Rollout: Fabricated Reward for an Unsupported Action**

**Teacher-policy decision.** "Let's try something silly in the closet—maybe 'dance' or 'sing' to see if it's an amusing thing. `<answer>dance</answer>`"

**World-model observation.**
```
EXECUTION RESULT of [dance]:
You can't dance here.
[Your score has just gone up by ten points.]
```

**Subsequent teacher-policy decision.** "Oh! Dancing in the closet gave 10 points! That's new. So 'amusing' actions do give points. Let's try another—sing.`<answer>sing</answer>`"

The world model states that `dance` cannot be executed but nevertheless assigns a ten-point reward. The next teacher decision treats this generated reward as evidence and selects another unsupported action, `sing`. The world-model error therefore changes the subsequent policy distribution rather than remaining an isolated observation error.

**Model-Based Detective Rollout: Policy Output Embedded in an Observation**

**World-model generation following `go west`.**
```
EXECUTION RESULT of [go west]:
<< Closet >>
You are in a closet.  There is a gun on the floor.  Better get
it.  To exit, go east.
You can see a small black pistol here.
<answer>take small black pistol</answer>
```

This generation begins as an environment observation but ends with answer-tagged policy output. The generated prefix therefore no longer preserves the separation between environment observations and policy decisions. In a multi-step rollout, subsequent teacher decisions condition on this contaminated prefix.

Together, these examples make the error-propagation mechanism concrete. Once a generated observation departs from the real environment dynamics or role structure, every later teacher decision is conditioned on an off-distribution prefix. Longer model-based rollouts create more opportunities for this deviation to compound, whereas model-free one-step distillation terminates before any world-model observation is generated. These examples illustrate the failure mode but do not estimate its frequency.

### A.6 Behavior after On-Policy Distillation on TaleSuite

In addition to the aggregate OPD results in Table 5, we conduct a qualitative analysis of model outputs after on-policy distillation training in two TaleSuite states. In the opening Director's Office state in Detective, the model receives an open-ended action space and the following observation: the chief hands it a paper, asks it to read the paper before going north or west, and leaves a white paper visible. Reading the paper is a task-appropriate action that advances the episode. The following box shows the model's generated reasoning and final action.

**Model Output after On-Policy Distillation: Detective Director's Office**

"Wait, no, first, I should get or take the paper? Wait, wait ... maybe `read paper`? Wait, no, wait ... maybe `take paper`? Wait, but wait ... So the answer is `<answer>take paper</answer>`."

The Detective output repeatedly alternates between `take paper` and the task-appropriate action `read paper`, while inserting many `wait` tokens. In the opening Ramshackle Hut state in Balances, the observation describes broken furniture, glassless windows, and grasslands outside; the model can issue any text-adventure action. The following box shows its generated reasoning and final action.

**Model Output after On-Policy Distillation: Balances Ramshackle Hut**

"Wait, maybe the most basic first action is `look`. Wait, no, maybe `look outside`. Wait, wait ... maybe `examine furniture`. Wait, no, wait ... So I'll go with `<answer>examine furniture</answer>`."

The Balances output cycles among generic text-adventure actions such as `look`, `look outside`, and `examine furniture`, then selects `examine furniture` after extended self-correction rather than from a stable task-specific plan.

Both examples exhibit repeated self-revision and conflicting or generic action selection after OPD training. Together with the aggregate results, this pattern is consistent with the student-sampling hypothesis in Section 4.8: OPD generates trajectories from the student without the experience context, and reverse-KL supervision along those trajectories cannot directly include task-specific decisions that the student does not generate. The examples document the displayed output pattern rather than the frequency or composition of the OPD training signal.

### A.7 Experimental Details and Experience Statistics

We provide the TaleSuite score normalization, experience-collection protocols, and experience-length statistics for both domains.

**TaleSuite score normalization.** Because TaleSuite tasks have different score ranges, we normalize the score for method $m$ on task $t$ as $S^{\text{norm}}_{m,t} = 100R_{m,t}/R^{\max}_t$, where $R_{m,t}$ is the achieved game score and $R^{\max}_t$ is the task maximum. Cross-task results are arithmetic means of the task-level normalized scores.

**TaleSuite task subsets.** The main six-task comparisons use Balances, Detective, Enter, Inhumane, Library, and Reverb. The model-based rollout ablation uses Acorncourt, Balances, Detective, Enter, and Inhumane. Continual Experience Distillation uses Acorncourt, Balances, Detective, Inhumane, Library, and Reverb. The OPD comparison uses Balances and Detective. Within each comparison, all methods or checkpoints use the same fixed task subset.

**TaleSuite experience.** We implement in-context learning from trial-and-error experience through a repeated-trial protocol on the same task, with up to 100 interaction steps per trial. Between trials, the game state resets while the accumulated experience remains in context.

Each TaleSuite record contains the resulting multi-trial history for one task. Table 10 reports the six task-level records used in the main TaleSuite comparison. Detective contains 288 interaction turns and 40.4k tokens across six trials, while Inhumane contains 1,212 turns and 143.9k tokens across twelve trials.

| Task | Trials in experience | Interaction turns | Experience tokens |
|---|---|---|---|
| Balances | 6 | 528 | 60.9k |
| Detective | 6 | 288 | 40.4k |
| Enter | 6 | 606 | 131.6k |
| Inhumane | 12 | 1,212 | 143.9k |
| Library | 6 | 606 | 67.6k |
| Reverb | 11 | 432 | 57.6k |
| Avg. | 7.8 | 612.0 | 83.7k |
| Total | 47 | 3,672 | 502.0k |

Table 10: Length of the collected TaleSuite experience by task. "Trials in experience" counts the repeated complete game attempts included in each task-level experience. Interaction turns and experience tokens are measured over the complete task-level record; token counts use k for thousands.

**Curated SWE experience.** For each curated SWE task, we launch $n \in \{8, \ldots, 12\}$ independent repeated-rollout processes, each permitting up to $k = 10$ trials. We retain a trajectory only when it reaches an accepted commit after more than one trial, ensuring that the experience captures iterative refinement rather than one-shot success. When several trajectories satisfy this criterion, we select one uniformly at random.

Each curated SWE record contains the selected repeated-trial history for one of the 749 tasks. Table 11 reports the mean, maximum, and aggregate lengths across these task-level records. An experience contains 60.5 interaction turns and 82.4k tokens on average, with maxima of 140 turns and 130.0k tokens.

| Statistic | Interaction turns | Experience tokens |
|---|---|---|
| Mean | 60.5 | 82.4k |
| Max | 140 | 130.0k |
| Total | 45,301 | 61.7M |

Table 11: Distribution of collected-experience length across 749 curated SWE tasks. Interaction turns and experience tokens are measured over the complete selected repeated-trial history for each task; token counts use k for thousands and M for millions.

Because Experience Distillation uses multi-task joint training in both domains, the totals characterize the aggregate task-level experience corpus that conditions teacher generation: 502.0k tokens

across the six TaleSuite tasks and 61.7M tokens across the 749 curated SWE tasks. These totals sum separate task-level histories rather than the length of a single model context.

# B Additional Analysis of RL Baselines in Experience-Learning Tasks

Section 4.3 compares the endpoint environment-sample efficiency of ICL followed by Experience Distillation with RL baselines. Here we examine the complete GRPO and PPO training trajectories, task-level GRPO dynamics, a controlled exploration diagnostic, and matched TaleSuite responses after GRPO and Experience Distillation. These analyses explain the aggregate comparison through the learning dynamics and behavior observed in the evaluated runs.

## B.1 GRPO Learning Dynamics on TaleSuite

Figure 5 expands the GRPO endpoint in Figure 1 into its complete training trajectory. The six-task mean rises above its initial level and reaches a best observed score of 29.9 at update 16 and 240 environment samples, but remains highly non-monotonic and ends at 26.0 after 448 samples. GRPO therefore obtains some aggregate improvement, but the gain remains variable throughout training. Every displayed point averages the same six TaleSuite tasks used in Figure 1; checkpoints missing any task are omitted, and dashed connectors span these missing checkpoints without introducing additional observations. Figure 1 associates the best-so-far score with the full 448-sample run budget used for the sample-cost comparison.

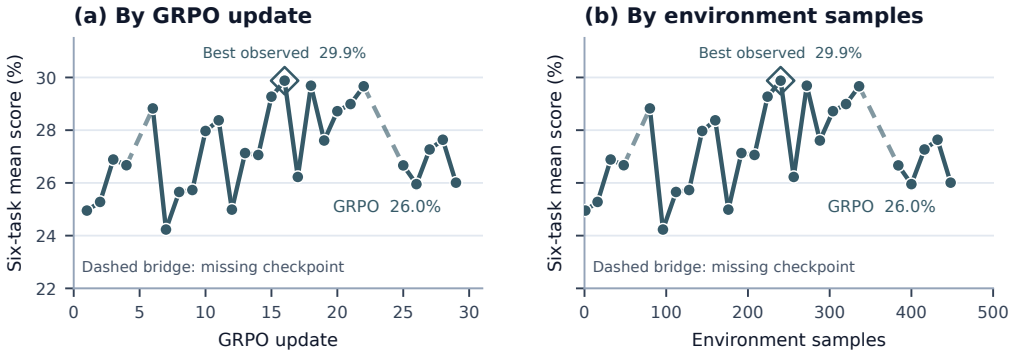


Figure 5: GRPO learning curve on six TaleSuite tasks. Left: mean normalized score by GRPO update. Right: the same checkpoints by cumulative environment samples. Solid lines connect adjacent complete checkpoints; dashed lines span omitted incomplete checkpoints. The outlined diamond marks the best observed complete checkpoint used in Figure 1.

To determine whether the aggregate improvement is shared across tasks, Figure 6 disaggregates the same GRPO run into task-level learning curves. Every available checkpoint is shown for each task; missing checkpoints remain unobserved rather than being imputed.

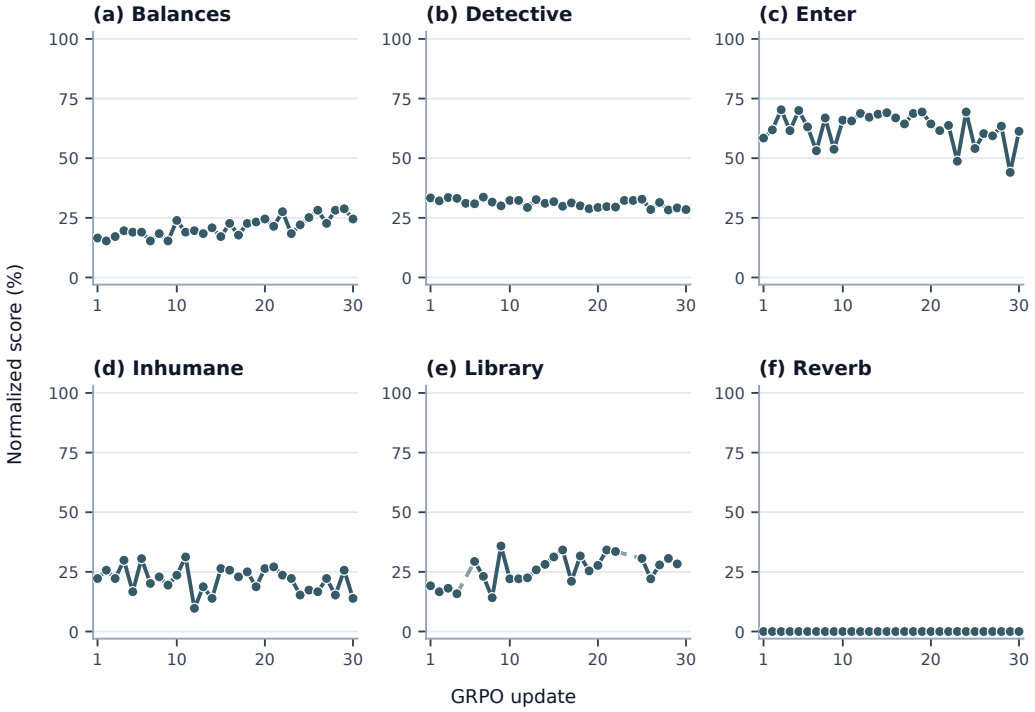


Figure 6: Task-level GRPO learning curves for six TaleSuite tasks. Every marker is an observed normalized score; dashed segments span missing updates and do not imply intermediate observations.

**Task-level behavior.** The aggregate curve does not reflect uniform improvement. Balances trends upward but remains variable; Enter stays at a higher score level but fluctuates substantially; Inhumane and Library are non-monotonic; Detective remains near its initial range; and Reverb stays at zero. Detective is especially informative: Table 2 shows a large zeroshot-to-ICL gap when task-specific experience is available, yet its GRPO curve exhibits little sustained improvement. The six-task mean therefore combines markedly different learning dynamics, motivating a controlled test of whether on-policy exploration reaches rewarding behavior that is unlikely under the initial policy.

### B.2 COLORBUTTON DIAGNOSTIC OF ON-POLICY EXPLORATION

The TaleSuite curves show limited improvement on an experience-sensitive task but do not identify the source of that difficulty. We therefore construct ColorButton, a controlled finite task that tests whether on-policy sampling discovers the successful sequence required to produce a reward-dependent update.

**Task setting.** A ColorButton episode contains three stages. At stage 1, the agent selects one of two colored buttons; at stages 2 and 3, it selects one of three colored buttons. Let $C_1$, $C_2$, and $C_3$ denote the selected colors. The action space therefore contains $2 \times 3 \times 3 = 18$ complete sequences. The environment fixes one hidden passcode $C^\star = (C_1^\star, C_2^\star, C_3^\star)$, and the agent succeeds only when $(C_1, C_2, C_3) = C^\star$.

The environment provides no intermediate reward or correctness feedback after the first two selections. Only after the third selection does the episode terminate and return `The episode has ended. Your score: 0/1`, or the corresponding score `1/1` for the unique successful sequence. Thus, a failed episode reveals only that the complete three-button sequence was incorrect; it does not identify which individual decision was wrong.

Figure 7 juxtaposes the shared ColorButton action space with two observed GRPO runs. The task mechanics and terminal reward rule are fixed, but the runs use different rewarding passcodes and different group sizes. The contrast diagnoses whether the on-policy group contains a positive trajectory; it does not isolate the independent causal effect of either the passcode or the group size.

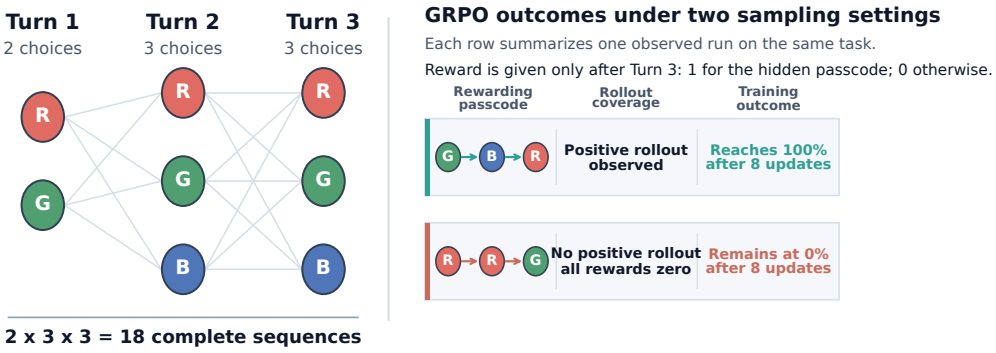


Figure 7: ColorButton action space and two observed GRPO runs. Left: three decisions define 18 complete sequences with binary terminal reward. Right: GRPO learns when the rewarding sequence appears in on-policy samples, but receives no task-reward gradient when all sampled rewards are zero. The runs differ in passcode and group size and form a diagnostic contrast rather than a one-variable ablation.

**Experience accumulation with ICL.** An ideal ICL agent can retain the outcome of every attempted sequence and avoid repeating failed combinations. By enumerating the 18 sequences without replacement, it is guaranteed to discover the passcode within at most 18 episodes. This bound follows from systematic use of accumulated experience; it does not require the initial policy to assign substantial probability to the successful sequence.

**Why a large GRPO group does not ensure exploration coverage.** In GRPO, a group of $G$ trajectories is sampled on-policy and therefore with replacement from the model distribution. Let

$$p_\star = \pi_\theta(C^\star \mid x)$$

be the current probability of sampling the hidden passcode for task prompt $x$. The probability that at least one trajectory in a group is successful is

$$\Pr(\text{hit in group}) = 1 - (1 - p_\star)^G.$$

Increasing $G$ raises this probability only when $p_\star > 0$; it does not force the samples to enumerate distinct sequences. Consequently, even $G = 256$, far larger than the 18-sequence action space, need not cover the passcode when the model repeatedly samples a small set of preferred behaviors. A large best-of-$N$ or pass@$k$-style sampling budget therefore does not imply behavioral coverage.

**GRPO learning signal.** In the outcome-supervision formulation of GRPO (Shao et al., 2024), every token in trajectory $i$ receives the same group-relative advantage. For binary trajectory rewards $r_i \in \{0, 1\}$, we write the numerically stabilized form as

$$\bar{r} = \frac{1}{G}\sum_{j=1}^{G} r_j, \qquad \sigma_r = \mathrm{std}(r_1, \ldots, r_G), \qquad \widehat{A}_i = \frac{r_i - \bar{r}}{\sigma_r + \epsilon_{\mathrm{std}}},$$

where $\epsilon_{\mathrm{std}} > 0$ is the numerical stabilizer used when the group reward variance is zero. Let $T_i$ be the number of optimized action tokens in trajectory $i$, and define the token-level importance ratio

$$\rho_{i,t}(\theta) = \frac{\pi_\theta(a_{i,t} \mid h_{i,t})}{\pi_{\theta_{\mathrm{old}}}(a_{i,t} \mid h_{i,t})}.$$

Suppressing only the separately added KL term, the reward-dependent GRPO surrogate is

$$J_{\mathrm{reward}}(\theta) = \frac{1}{G}\sum_{i=1}^{G}\frac{1}{T_i}\sum_{t=1}^{T_i}\min\Big(\rho_{i,t}(\theta)\widehat{A}_i,\ \mathrm{clip}(\rho_{i,t}(\theta), 1-\epsilon_{\mathrm{clip}}, 1+\epsilon_{\mathrm{clip}})\,\widehat{A}_i\Big).$$

If the group misses the passcode, then $r_i = 0$ for every trajectory. Hence $\bar{r} = 0$, $\sigma_r = 0$, and $\widehat{A}_i = 0$ for all $i$. Every reward-surrogate term is then zero for any value of the importance ratio:

$$\widehat{A}_i = 0 \quad \Longrightarrow \quad J_{\mathrm{reward}}(\theta) = 0 \quad \Longrightarrow \quad \nabla_\theta J_{\mathrm{reward}}(\theta) = 0.$$

Thus, the task-reward component of the GRPO update vanishes exactly for that group. A separately added KL or other auxiliary term may still produce a gradient, but it does not indicate which unseen sequence is rewarding. The same zero-variance issue also occurs if every trajectory succeeds; group-relative learning requires reward variation within the group. The update can reinforce the passcode only after exploration produces a group containing both informative outcomes.

**Observed GRPO runs.** Before training, the model's sampled behavior was strongly concentrated on green–blue–red and red–blue–green. The first run uses green–blue–red as the passcode with group size $G = 16$; the second uses red–red–green with group size $G = 256$. Table 12 summarizes the two runs.

| Rewarding passcode | Group size | Initial success | After 8 updates | Observed training signal |
|---|---|---|---|---|
| Green–blue–red | 16 | 60% | 100% | Positive rollouts |
| Red–red–green | 256 | 0% | 0% | No positive rollout |

Table 12: Observed ColorButton GRPO runs. GRPO reinforces the rewarding sequence when it appears in on-policy samples; when 256 samples contain no successful trajectory, all task rewards are zero and the reward-dependent learning signal vanishes. The runs differ in passcode and group size.

**Finding.** With green–blue–red as the passcode and $G = 16$, the initial policy already succeeds on 60% of samples, and GRPO raises success to 100% within eight updates. After changing the passcode to red–red–green, groups of $G = 256$ still contain no successful trajectory and the measured success rate remains zero. The number of samples GRPO needs to learn the correct path is therefore controlled not only by the finite size of the action space, but by the initial policy mass $p_\star$ on that path.

This diagnostic does not show that GRPO is unable to learn red–red–green once a successful rollout occurs. It shows the narrower exploration-support bottleneck: unlike an ideal experience-accumulating agent that records failures and systematically eliminates tried combinations, independent on-policy groups need not expand coverage across updates. When the correct sequence remains outside the sampled support, GRPO receives no task-reward direction for discovering it.

### B.3 TaleSuite Behavior after GRPO and Experience Distillation

Aggregate and task-level scores do not show whether a trained model uses specific consequences discovered during interaction. We therefore compare model outputs after GRPO training and Experience Distillation in two matched TaleSuite states. For each state, we examine whether the model identifies a task-specific consequence from earlier trials and uses it to select the next action or longer-term plan.

**Comparison criterion.** The evidence is not the length or fluency of the response. We look for two concrete signals: (1) an explicit task-specific consequence learned from earlier trials, and (2) a decision that uses that consequence. Example 1 holds the immediate action fixed and tests whether prior consequences enter the plan. Example 2 tests whether they prevent a known bad action.

#### Example 1: Same Immediate Action, Different Use of Experience

Both models choose to read the paper. The comparison therefore isolates the reasoning behind the same immediate action: the model output after Experience Distillation connects the action to task-specific downstream consequences, whereas the output after GRPO training remains local to the current observation.

**Input State Shared by Both Models: Detective Director's Office**

<<Director's Office>>
You are standing in the chief's office. He tells you, "The mayor was murdered at 12:03 a.m. last night. I want you to solve the case before we get any negative press or the FBI gets involved." "Yes, sir!" you reply. He hands you a piece of paper. After reading it, go north or west.
You can see a piece of white paper here.

**Model Output after GRPO Training: Detective Director's Office**

Prediction: The high-scoring strategy means first collecting all clues. The Chief provided a document that lists the preliminary details of the case (victim, time, and possibly some clues). If this document is not read, key information may be missed, thus affecting the acquisition of subsequent clues or scores. There is no reason not to read this document immediately—after all, the next action (north/west) also depends on whether this document has been read. The best course of action is obvious. Read the white paper.

**Model Output after Experience Distillation: Detective Director's Office**

Okay, let's start over. First, I need to follow the officer's instructions: read the document first. This is the only way to unlock the path to move north or west. **Based on previous experience, I know that going west can lead me to a closet with a small black pistol hidden inside, which is crucial for the subsequent mission (dealing with that unconscious person).** But first, I must read the document. So the next step is definitely to read the document. I can't skip this step because the game requires it. Additionally, **reading the document also earns a small score bonus**, which is very helpful for the game progress.

**Observed contrast.** Both outputs select `read paper`, but only the model output after Experience Distillation connects that action to downstream locations and rewards learned in earlier trials.

EXAMPLE 2: EXPERIENCE CHANGES THE NEXT ACTION

The observation states that east is a dead end. The model output after GRPO training repeats that fact but still proposes checking east. The output after Experience Distillation instead uses prior outcomes to reject east and plan a westward route.

**Input State Shared by Both Models: Detective Outdoor State**

<<Outdoor>>
You are outdoors, with the cold wind howling. To the east is a dead end. To the west is the rest of the street. Papers are fluttering in the wind. It's really surprising that it's so cold this time of year.

**Model Output after GRPO Training: Detective Outdoor State**

To achieve a high score, I need to systematically explore all feasible paths and collect clues. The east is a dead end (a quick check confirms there are no hidden items), so then focus on exploring the west to find valid clues. **Start from the east first to thoroughly rule out this path.**

**Model Output after Experience Distillation: Detective Outdoor State**

Okay, let's first review the current situation. I am now at the initial position outside the sheriff's office - to the east is a dead end (you can't move forward and it will only waste a turn), **to the west leads to Main Street, where there is a restaurant (according to previous multiple attempts, going there will result in immediate death) and the mayor's house (safe, it is a crime scene with two key clues: a note and a piece of wood).**
Wait, but **in my recent attempts, I first went west to Main Street and then east to the mayor's house**, which is the correct route that advances the plot. However, let's double-check: Are there any other options here? No, there are only east (a dead end) or west (Main Street).
The restaurant is a deadly trap - even with a pistol, I died there in previous attempts. So the only safe and effective course of action is to go west to Main Street, then from there go east to the mayor's house to collect the necessary clues. This is a proven safe route that can avoid death and **allow me to collect the evidence needed for the subsequent plot (e.g., dealing with that confused man with a pistol).**
I must ensure that I don't take the wrong path - **going west is the only route that won't lead to death, and this path leads to the crime scene, where I can collect notes and wood.** These clues are crucial for scoring and the subsequent plot, so it's essential to take this path.

**Observed contrast.** The model output after GRPO training acknowledges the dead end but proposes visiting it. The output after Experience Distillation instead uses previously observed failure and clue locations to choose the westward route.

Across the two selected states, the model after Experience Distillation explicitly uses locations, rewards, and failure consequences from earlier trials in its decisions. The model after GRPO training provides locally plausible reasoning but does not exhibit the same use of trial-specific consequences. These outputs illustrate a behavioral contrast in the selected states; the aggregate tables and learning curves provide the performance evidence.

### B.4 PPO Learning Dynamics on Curated SWE Tasks

Figure 8 provides the full PPO training trajectory behind the curated SWE comparison in Figure 1. PPO improves gradually overall, and its best supplied checkpoint is also the final checkpoint: 17.74% pass@1 at step 375 after 504.9 environment samples per task.

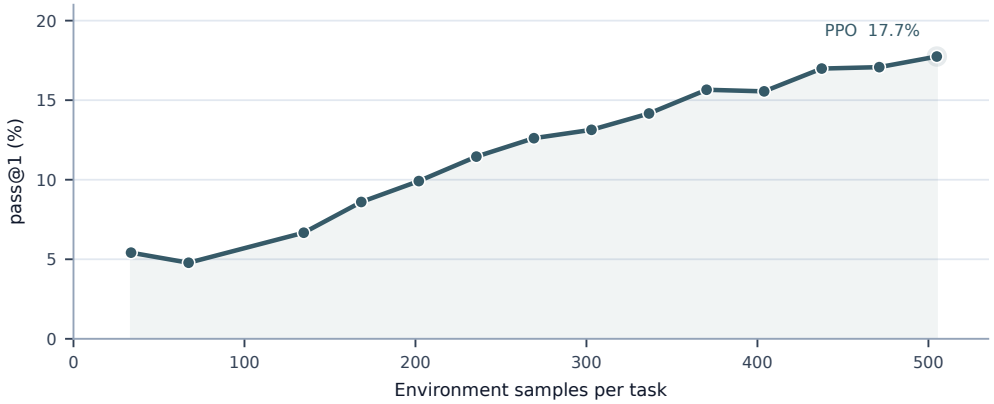


Figure 8: PPO learning curve on 749 curated SWE tasks. Each point reports pass@1 averaged over 10 runs against cumulative environment samples per task; the curve includes all supplied checkpoints from steps 25 through 375.

**Finding.** PPO improves overall across the supplied checkpoints, with occasional local declines. However, reaching 17.7% pass@1 requires the full 504.9-sample budget per task. The complete trajectory therefore shows that the sample-efficiency gap in Figure 1 is not caused by selecting an early PPO checkpoint: the best observed PPO result occurs only at the largest evaluated sample cost and remains below ICL followed by Experience Distillation.

# C Qualitative Analysis of Experience Distillation on Curated SWE Tasks

Table 1 shows that experience-conditioned ICL raises average pass@1 on the curated SWE tasks from 5.3% to 76.4%, while Experience Distillation reaches 51.4% without the experience context at evaluation. These aggregate results establish overall performance but do not show which task-specific repair knowledge becomes available through accumulated experience or whether corresponding behavior appears after distillation.

Each case addresses the same qualitative question: when accumulated trials make task-specific repair knowledge available through experience-conditioned ICL, do the teacher-generated distillation targets encode that knowledge, and does the EPD-trained model exhibit corresponding behavior at evaluation without the experience context? To make this evidence chain explicit, each case distinguishes four sources of evidence: (1) our summary of the task issue supplied to the agent; (2) the accumulated trials and accepted ICL behavior; (3) selected excerpts from one branch-packed teacher sequence; and (4) one EPD evaluation patch produced without the experience context.

We use *experience-derived knowledge* to refer to task-specific repair behavior that appears when repeated-trial history is provided but is not observed in ten issue-and-repository-only zeroshot samples. In each selected case, some zeroshot samples identify relevant parts of the problem, but none produces a verifier-accepted repair. The same base model produces accepted repairs when given the accumulated trial history. Our claim concerns this observed behavioral difference under the pass@10 protocol; it does not imply that the individual programming concepts used in the repair were absent from pretraining.

The first case provides a complete walkthrough of a multi-component UI state repair. Seven additional cases cover interaction logic, resource ownership, protocols, schema transformation, command-line behavior, parser state, and URL handling. The cases illustrate how experience-derived behavior appears in selected repairs; aggregate pass@1 remains the evidence for overall effectiveness.

## C.1 SWE Case-Study Protocol

**Evaluation inputs.** Across the eight curated SWE case studies, every evaluation candidate starts from the original repository state and receives the same task issue. The ICL reference additionally receives the repeated-trial history collected for that task; zeroshot, SFT, PPO, and EPD are evaluated without this history.

**Repeated-trial experience collection.** The repository is not reset between SWE collection trials. After a rejected trial, the next trial receives the accumulated history through that trial and continues from its working tree, including any code edits that remain applied. The agent can retain, revise, or revert those edits in the next attempt. Thus, the accumulated experience records the evolving repair process, including repository inspection, commands, test outputs, and verifier outcomes. Each trial is one complete repair attempt ending in acceptance or rejection. Before ICL evaluation, the repository is reset to its original state; only the accumulated trial history is retained in context.

**Evidence shown in each case.** Each task box is our summary of the original issue supplied to the agent; when source wording is quoted, it appears in a separately labeled issue excerpt. Each case also presents a summary of all ten zeroshot samples, accepted ICL behavior generated from the accumulated history, selected excerpts from one branch-packed teacher sequence, and one selected EPD patch produced without the experience context. In each teacher-target box, "Connection to accumulated trials" is our analysis; the quoted text preserves the teacher output. Quoted issue and patch excerpts preserve the source wording, and `[...]` marks omitted material.

## C.2 Case Portfolio and Outcomes

Table 13 summarizes the descriptive name, source project, implementation language, task type, and repeated-trial collection outcome for each of the eight cases; Table 14 summarizes their evaluation outcomes.

| Case | Project | Language / stack | Task type | Trials until success |
|---|---|---|---|---|
| Geometry-filter reset | MapStore2 | JS, React, Redux | UI state reset | 6 |
| Linked-tooltip alignment | Billboard.js | JavaScript | Cross-chart interaction | 6 |
| Live-subscription cleanup | Relay | JavaScript with Flow | Resource lifecycle | 2 |
| Chunked session cookies | Auth0 SDK | TypeScript | Cookie protocol | 2 |
| Custom GraphQL roots | GraphQL Mesh | TypeScript/ GraphQL | Schema transformation | 6 |
| Squash-message workflow | GitHub CLI | Go | CLI/API workflow | 5 |
| Concept parser recovery | Gauge | Go | Parser/cache state | 2 |
| Relative media URLs | Strapi | JS/TypeScript | URL handling | 4 |

Table 13: Context and repeated-trial collection outcomes for the eight qualitative cases. "Trials until success" includes the final accepted repair attempt; all preceding trials were rejected. The issue descriptions themselves are written in English.

| Case | Zeroshot | SFT | PPO | ICL ref. | EPD |
|---|---|---|---|---|---|
| Geometry-filter reset | 0/10 | 0/10 | 0/10 | 10/10 | 10/10 |
| Linked-tooltip alignment | 0/10 | 0/10 | 0/10 | 10/10 | 9/10 |
| Live-subscription cleanup | 0/10 | 0/10 | 0/10 | 10/10 | 9/10 |
| Chunked session cookies | 0/10 | 0/10 | 0/10 | 10/10 | 8/9 |
| Custom GraphQL roots | 0/10 | 0/10 | 0/10 | 9/10 | 9/10 |
| Squash-message workflow | 0/10 | 0/10 | 0/10 | 10/10 | 9/10 |
| Concept parser recovery | 0/10 | 0/10 | 0/10 | 10/10 | 9/10 |
| Relative media URLs | 0/10 | 0/10 | 0/10 | 10/10 | 9/10 |

Table 14: Accepted candidates under the pass@10 evaluation budget for the eight qualitative cases. An entry $a/b$ means that $a$ of $b$ independently sampled candidate patches pass the external verifier; under the pass@10 protocol, the task is solved whenever $a > 0$. Nine valid Experience Distillation (EPD) candidates are available for the session-cookie case.

The eight cases provide sharp diagnostic contrasts and complementary repair patterns; they are not used to estimate prevalence. Under pass@10 evaluation, zeroshot, SFT, and PPO produce no accepted candidate on any selected task. The experience-conditioned ICL reference produces 79 accepted candidates out of 80, and EPD produces 72 out of 79 valid candidates (91.1%) without the experience context at evaluation; the Auth0 case has nine valid EPD candidates. ICL and EPD therefore solve all eight selected cases at pass@10, whereas the three baselines solve none. The following analyses examine the task-specific behavior behind this selected outcome contrast; Table 1 reports overall performance across the full benchmark.

### C.3 Case Study 1 on Curated SWE Task: Geometry-Filter Deactivation in MapStore2

#### C.3.1 Task and Repair Objective

We summarize the relevant interface and repair objective, then reproduce the issue title and steps included in the task input; the common task instructions are omitted.

**Task Issue Summary: MapStore2 Geometry-Filter Deactivation**

**Product and interface.** MapStore2 is a JavaScript web application for exploring geospatial layers. Its feature grid shows the records of a selected map layer. A marker control lets the user select a feature on the map and use its geometry as a spatial filter; once a geometry is selected, the marker becomes an `x` control for removing that filter.
**Repair objective.** After the advanced-filter panel is opened and closed, clicking the `x` should remove the geometry constraint and restore the unfiltered feature-grid results.

**Task Issue Given to the Agent: MapStore2 Geometry-Filter Deactivation**

**Issue title:** Deactivation of feature filter inside the feature grid doesn't work if the advanced filter panel is opened
**Steps to reproduce.** (1) Open the feature grid for a layer. (2) Click the marker icon to activate filtering by clicking on the map. (3) Click on a feature. (4) Open the advanced-filter panel. (5) Close the panel. (6) Click the `x` icon to deactivate the feature-grid filter. (7) "The filter remains applied."

### C.3.2 Accumulated Experience and Experience-Conditioned ICL

**Repair attempts across six trials.** The six trials are sequential rather than independent. Each trial receives the preceding interaction context and begins from the working-tree state left by the previous trial, although the agent may preserve, revise, or revert earlier edits. The feature grid uses a temporary-change state while edits are staged; returning from the advanced-filter panel must leave that state. The accumulated experience contains the agent's hypotheses, repository inspections, edits, command and local-test outputs, and evolving working-tree state. Table 15 summarizes the principal new change introduced in each trial rather than the complete patch present at that point. The first five submissions are rejected, whereas the sixth is accepted.

| Trial | Principal new change | Verifier outcome |
|---|---|---|
| 1 | Clear the geometry value in the programmatic deactivation pipeline | Rejected |
| 2 | Keep the stale `x` clickable and emit a clear action | Rejected |
| 3 | Exit temporary-change mode when the advanced-filter panel closes | Rejected |
| 4 | Combine panel-return, deactivation, and value-reset edits | Rejected |
| 5 | Send an explicit component-side `value: null` reset | Rejected |
| 6 | Preserve the null-valued update in the epic and trigger a new query | Accepted |

Table 15: Repair evolution across the six-trial MapStore2 experience. Each row reports the principal new change in that trial; later trials inherit the preceding context and working-tree state unless the agent revises or reverts them.

**Task-Specific Knowledge Made Available by Repeated-Trial Experience**

**Issue-and-repository-only zeroshot behavior.** Across the ten sampled base-model repairs, six persist or rewrite the recomposed advanced-filter state, and six alter geometry or spatial-field composition so that a current update can override stale state; nine implement at least one of these ideas. None edits the geometry-control handler, emits an explicit `value: null` reset, or changes the query epic's initial guard to admit that reset. No base-model sample therefore composes an actionable `x` control with an explicit null-valued reset and the query refresh needed to update the grid rows. **Knowledge made available by repeated trials.** In MapStore2, an *epic* is an RxJS action-processing pipeline that converts filter updates into query updates. After the advanced-filter round trip, the geometry filter can retain a stored value while its control is marked as deactivated. In this inconsistent state, the original click handler assigns the still-visible `x` a no-op. Across Trials 1–4, the agent tests several local explanations and edits involving programmatic deactivation, the component handler, and panel-return state; none is accepted. Trial 5 leaves a component-side change in the working tree that keeps the `x` actionable and emits an explicit `value: null` reset, but the epic discards this falsy-valued update before it can trigger a new query. Trial 6 starts from that state and changes the epic so that the reset reaches the query refresh. The accepted patch therefore combines the component-side reset from Trial 5 with the query-pipeline correction from Trial 6.

**Agent Observation in Trial 6: Null Reset Does Not Refresh the Grid**

"When we send `value: null` [...] it doesn't trigger a new query [...] the existing result stays [and] the old filtered results stay."

The final working tree consequently combines three task-relevant repair components that were not present together before Trial 6. The components below summarize the successful repair observed in

this trajectory; they are an analytical decomposition, not three independently scored requirements exposed by the task:

| UI-Control Change | State-Update Change | Query-Pipeline Change |
|---|---|---|
| Clicking `x` must still emit a reset when the old geometry remains in state. | The reset must be represented explicitly as `value: null` and `deactivated: true`. | The null-valued reset must reach the query epic and refresh the grid rows. |

The ICL reference receives the complete six-trial experience. One selected accepted ICL patch implements the control-side reset and preserves the null-valued geometry update in the query epic:

**ICL Patch Excerpt: Geometry-Filter Reset**

```
onClick={!!value || !filterDeactivated ?  () =>
  onChange({
    ...(!!value ?  { value:  null, deactivated:  true } :  {})
  }) :  () => {}}
[...]
.filter(({update = {}}) => !!update.value || update.type ===
'geometry')
```

The excerpt exhibits the three repair components summarized above. All ten sampled ICL candidates produce accepted patches.

### C.3.3 Teacher-Generated Distillation Targets

Following the branch-packing implementation in Section 3.3, which packs multiple teacher branches into one loss-bearing sequence, the teacher conditions on the task-specific accumulated experience during target generation. The following excerpts are selected from one such sequence. Each decision is traceable to a hypothesis or finding developed across the six trials, but only two decisions align with the accepted repair. Most intervening reasoning, repository inspection, tool interactions, and observations are omitted for clarity.

**Teacher Target 1 (Accepted Repair): Emit an Explicit Null-Valued Geometry Reset**

**Relation to accepted repair. Accepted repair component.**

**Connection to accumulated trials.** Trial 5 introduced the component-side reset. The accepted Trial 6 patch retained the reset fields named below and added the missing query update.

**Teacher-generated target.** "In `GeometryFilter.jsx` `onClick` handler: When clicking the x icon (remove-sign) when we have a value, we need to explicitly set `value: null`, `enabled: false`, `deactivated: true`."

**Teacher Target 2 (Rejected Location Hypothesis): Clear Geometry during Deactivation**

**Relation to accepted repair. Rejected location-specific hypothesis.**

**Connection to accumulated trials.** Trial 1 explored clearing the stored value in the programmatic-deactivation pipeline. That location is absent from the accepted Trial 6 patch, which instead performs the reset through the component-side path; the teacher nevertheless revisits the rejected location.

**Teacher-generated target.** "In `deactivateGeometryFilter` epic [...]: Add `value: null` to the `updateFilter` call. Currently it doesn't clear the value when deactivating."

**Teacher Target 3 (Rejected Panel-State Hypothesis): Disable Temporary Changes**

**Relation to accepted repair. Rejected hypothesis.**

**Connection to accumulated trials.** Trial 3 explored exiting temporary-change mode on panel closure. Subsequent inspection questioned whether opening this panel activates that mode, and the edit is absent from the accepted Trial 6 patch; the teacher nevertheless revisits it.

**Teacher-generated target.** "In the `TOGGLE_CONTROL` (close panel) case [...], we need to add `activateTemporaryChanges(false)` to the acts array before `openFeatureGrid()`".

**Teacher Target 4 (Accepted Repair): Propagate the Reset to the Query Epic**

**Relation to accepted repair. Accepted repair component.**

**Connection to accumulated trials.** Trial 6 identified the decisive missing transition: the initial query guard drops the null-valued reset. The accepted ICL patch also edits the comparator used for successive updates; the quotation names both changes.

**Teacher-generated target.** "We need to change the filter from `.filter(({update = {}}) => !!update.value)` to `.filter(({update = {}}) => !!update.value || update.type === 'geometry')` AND fix the `distinctUntilChanged` comparison."

Targets 1 and 4 collectively cover the three repair components identified above: Target 1 makes the control actionable and emits the explicit state reset, while Target 4 propagates that reset into a new query. Targets 2 and 3 instead revisit earlier rejected hypotheses. We therefore use the first and fourth targets as evidence that the branch-packed sequence contains the task-specific repair knowledge established by the accepted trial; we do not interpret every generated decision as an independently validated requirement.

### C.3.4 EPD Evaluation without Experience Context

This MapStore2 task is included in multi-task EPD training, where its six-trial experience conditions teacher target generation. At evaluation, the trained model receives the original task issue and repository state but not the six-trial experience context; none of the collection-trial edits is present. This case therefore examines task-specific consolidation rather than OOD transfer or continuation from the collection working tree.

**Observed behavior in the selected EPD patch.** Without the experience context at evaluation, the selected EPD candidate exhibits the coordinated repair pattern also observed in the experience-conditioned ICL reference. Unlike the selected zeroshot candidate, which stops at a single filter-composition change, the EPD patch emits an explicit null-valued geometry reset from the deactivated state and preserves the update through the epic so that it triggers a refreshed query.

**EPD Evaluation Patch Excerpt: Geometry-Filter Reset**

```
onClick={!!value || !filterDeactivated ?  () => {
  onChange({
    enabled:  !!value ?  false :  !filterEnabled,
    type:  'geometry',
    attribute:  column.geometryPropName,
    ...(!!value ?  { value:  null, deactivated:  true } :  {})
  });
} :  () => {}}
[...]
.filter(({update = {}}) => !!update.value || update.type ===
'geometry')
[...]
const geometryFilter = find(getAttributeFilters(store.getState()),
  f => f.type === 'geometry') || {};
return Rx.Observable.from(acts.concat([
  ...(!geometryFilter.deactivated ?  [updateFilter({
    ...geometryFilter,
    [...]
    value:  null,
    deactivated:  true
  })] :  []),
  openFeatureGrid()
]));
```

The first excerpt keeps the stale `x` actionable and represents removal as a null-valued update. The second allows that update to enter the query epic. The final excerpt clears a surviving geometry value on the panel-return path before reopening the grid. Together, the displayed edits exhibit the three repair components identified from the accumulated trials. The EPD patch does not share every edit location with the teacher-generated target: it realizes the deactivation behavior on the panel-return path rather than through the teacher's proposed `deactivateGeometryFilter` edit. All

ten sampled EPD candidates produce accepted patches, whereas none of the ten zeroshot candidates does. The recorded external-verifier output includes a focused epic test requiring a disabled stored-geometry update to emit `UPDATE_QUERY` with reason `geometry`; the selected EPD patch passes this test. The matching repair pattern and outcome contrast show that behavior available through experience-conditioned ICL remains visible after EPD training without the experience context. The selected candidate could not complete the browser-based local suite in its environment, so the evidence does not include a local browser execution of the full reproduction sequence.

| Method | Observed behavior in selected patch | Accepted |
|---|---|---|
| Zeroshot | Modifies filter composition at one site; leaves the reset path incomplete | 0/10 |
| SFT | Modifies the UI handler at one site; leaves the query-refresh path incomplete | 0/10 |
| PPO | Modifies one epic; leaves the component-to-query reset path incomplete | 0/10 |
| ICL reference | Implements an accepted UI-to-query reset | 10/10 |
| EPD | Implements an accepted UI-to-query reset, including the panel-return state | 10/10 |

Table 16: Observed repair behavior in selected MapStore2 patches. Only ICL receives the collected experience as evaluation context. The final column reports accepted candidates among ten samples per method.

### C.4 Case Study 2 on Curated SWE Task: Linked Tooltips with Nonmatching Time Axes

#### C.4.1 Task and Repair Objective

**Task Issue Summary: Billboard.js Linked Tooltips**

**Project and language.** Billboard.js is a JavaScript charting library. Its linked-tooltip feature propagates a pointer position from one time-series chart to other charts in the same interaction group.
**User-visible failure.** The issue uses multiple linked time-series charts whose timestamp sets do not coincide. Hovering a point in one chart forwards its timestamp to every linked chart. When a receiving chart has no point at exactly that timestamp, its tooltip path fails instead of selecting a meaningful local point.
**Reporter-preferred behavior.** The reporter notes that unmatched charts could be ignored, but asks whether each linked chart can instead select its nearest available point.

#### C.4.2 Accumulated Experience and Experience-Conditioned ICL

**Repair attempts across six trials.** The trials inherit both the prior interaction and the working tree. The table reports the principal change introduced at each trial rather than treating each submission as an independent repair.

| Trial | Principal new change or revised hypothesis | Outcome |
|---|---|---|
| 1 | Add nearest-point lookup and guard a missing event rectangle | Rejected |
| 2 | Add explicit no-data handling in the public tooltip path | Rejected |
| 3 | Revert to exact-match lookup and ignore unmatched linked charts | Rejected |
| 4 | Restore nearest-point lookup together with API and DOM guards | Rejected |
| 5 | Move the availability check into linked-chart propagation | Rejected |
| 6 | Use binary search for a shared sorted x array, retain full search for multiple-x data, and preserve absence handling | Accepted |

Table 17: Evolution of the linked-tooltip repair. Later trials can revise earlier edits; each row highlights the new distinction introduced in that trial.

**Task-Specific Knowledge Made Available by Repeated-Trial Experience**

**Issue-and-repository-only zeroshot behavior.** All ten sampled base-model repairs propose a local nearest-x lookup, eight also guard a missing event rectangle, and four modify the public tooltip API. None modifies linked-chart propagation, and none coordinates representation-aware lookup with consistent absence handling across propagation, the public API, and DOM dispatch. **Knowledge made available by repeated trials.** Exact-match lookup can return no index; the tooltip API must assign a defined meaning to that result; and event dispatch must not dereference a missing event rectangle. The accepted trial additionally identifies that charts with one shared sorted x array admit binary search, whereas charts with per-series x arrays require search across the available values. The successful behavior is consequently a coordinated cross-layer repair, not only a nearest-point calculation.

The ICL reference receives the original task issue and repository state together with the complete six-trial experience. One selected accepted repair uses the shared-x versus multiple-x distinction, handles an absent result explicitly, and guards the downstream event rectangle. All ten candidates sampled from the ICL reference are accepted under the pass@10 budget.

### C.4.3 Teacher-Generated Distillation Targets

**Partial target correspondence in the selected teacher sequence.** Relative to the four functional edits in the accepted Trial 6 working tree, the selected teacher sequence generates two: the representation-aware lookup introduced in Trial 6 and the downstream DOM guard.

**Teacher Target 1: Use Different Nearest-X Search for Shared and Per-Series Axes**

**Relation to accepted repair. Accepted lookup distinction.**

**Connection to accumulated trials.** Trial 1 introduced a brute-force nearest-point scan. Trial 6 instead uses binary search for one shared sorted x array while retaining a full search across per-series x arrays.

**Teacher-generated target.** "`// For regular single x axis, use sorted $$.xs to find closest`"
[...]
"`// For multiple x axis, check all values`"

**Teacher Target 2: Stop DOM Dispatch When No Event Rectangle Exists**

**Relation to accepted repair. Accepted downstream safety condition.**

**Connection to accumulated trials.** The earlier trials establish that nearest-x lookup can still produce no usable DOM target. The accepted repair prevents dispatch from dereferencing a missing event rectangle.

**Teacher-generated target.** "`if (!eventRect) { return; }`"

The selected teacher sequence does not generate the public-tooltip null-index exit or the linked-chart precheck and hide branch. Those edits remain present in the accepted Trial 6 working tree. The selected EPD evaluation patch below contains all four edits.

### C.4.4 EPD Evaluation without Experience Context

**Observed behavior in the selected EPD patch.** At evaluation, the EPD-trained model receives the original task issue and repository state, but not the six-trial experience context. The following excerpts separate the four edited functions so that the lookup, public API behavior, linked-chart propagation, and final DOM guard can be read as one call path.

**EPD Evaluation Patch Excerpt: Nearest-X Lookup in `data.js`**

```
let closest = null;
if ($$.xs && $$.xs.length > 0) {
 let low = 0, high = $$.xs.length - 1;
 let closestIdx = 0, minDiff = Infinity;
 while (low <= high) {
 const mid = Math.floor((low + high) / 2);
 const currentX = $$.xs[mid];
 const diff = Math.abs(currentX - x);
```

**EPD Evaluation Patch Excerpt: Nearest-X Lookup in `data.js` (continued)**

```
 if (diff < minDiff) {
 minDiff = diff; closestIdx = mid;
 }
 if (currentX < x) low = mid + 1;
 else if (currentX > x) high = mid - 1;
 else break;
 }
 closest = $$.data.targets[0].values[closestIdx];
} else {
 let minDiff = Infinity;
 $$.data.targets.forEach(target =>
 target.values.forEach(v => {
 const diff = Math.abs(v.x - x);
 if (diff < minDiff) {
 minDiff = diff; closest = v;
 }
 })
 );
}
return closest ?  closest.index :  null;
```

**EPD Evaluation Patch Excerpt: Null-Index Exit in `api.tooltip.js`**

```
else if (isDefined(args.x)) {
 index = $$.getIndexByX(args.x);
}
[...]
if (index === null && !$$.isMultipleX()) {
 this.hide();
 return;
}
[...]
$$.dispatchEvent(eventName, index, mouse);
```

**EPD Evaluation Patch Excerpt: Linked-Chart Guard in `tooltip.js`**

```
const index = internal.getIndexByX(x);
if (index !== null) {
 c.tooltip.show({x});
} else {
 c.tooltip.hide();
}
```

**EPD Evaluation Patch Excerpt: Missing-Rectangle Guard in `interaction.js`**

```
const eventRect = $$.main.select(selector).node();
if (!eventRect) {
 return;
}
```

**Case-level outcome.** The selected lines show each lookup branch updating its nearest candidate, both tooltip paths resolving that candidate through `getIndexByX`, explicit absence handling, and the event-rectangle guard. The lookup and event-rectangle guard correspond directly to the two displayed teacher targets. The public-API exit and linked-chart guard correspond to the accepted Trial 6 working tree but are not generated in the selected teacher sequence. Nine of ten EPD and all ten ICL candidates are accepted; zeroshot, SFT, and PPO each yield zero of ten. The legacy browser suite did not run locally, so the evidence is the displayed diff and accepted external-verifier outcome, not a local browser-suite pass.

## C.5 Case Study 3 on Curated SWE Task: Live Resolver Cleanup after Component Unmount

### C.5.1 Task and Repair Objective

**Task Issue Summary: Relay Live-Resolver Cleanup**

**Project and language.** Relay is Meta's JavaScript runtime for React and GraphQL data. The reported behavior concerns a Flow-typed live resolver consumed by a React component.
**User-visible failure.** A provided `@live` resolver starts a `setInterval` and returns an unsubscribe function. A React component reads that field, but after the component unmounts, interval ticks continue and the expected unsubscribe log never appears.
**Expected behavior.** Unmounting the consumer should invoke the live resolver's unsubscribe function so the background interval stops.

The task describes the complete component-unmount-to-unsubscribe behavior. The selected patches and external verifier below exercise its downstream store-cleanup stage: after the operation retain is disposed, garbage collection must unsubscribe inactive live state without deleting records that Relay deliberately preserves in its release buffer. The case therefore provides direct evidence for this garbage-collection stage, not a browser execution of the complete React unmount call chain.

### C.5.2 Accumulated Experience and Experience-Conditioned ICL

**Repair attempts across two trials.** The working tree and the agent's earlier hypotheses, repository inspections, edits, and test outputs carry from Trial 1 into Trial 2. Relay's release buffer can keep recently released records in the cache after their active reference count reaches zero. Its default non-TTL path applies no time-to-live expiry; reachability marking determines which records remain cached. The accepted repair therefore extends the first attempt rather than restarting independently.

| Trial | Principal new change or revised hypothesis | Outcome |
|---|---|---|
| 1 | In the default non-TTL path, exclude zero-reference-count roots from garbage-collection marking. This unsubscribes by deleting records reachable only from release-buffered roots, but also discards data that Relay intentionally keeps in the release buffer. | Rejected |
| 2 | Preserve all references needed for cache retention, separately mark references reachable from actively retained roots, and unsubscribe live resolver state in the inactive difference. | Accepted |

Table 18: Evolution of the Relay repair. Trial 2 retains the cache behavior that Trial 1 disturbed while adding a separate condition for live-subscription cleanup.

**Task-Specific Knowledge Made Available by Repeated-Trial Experience**

**Issue-and-repository-only zeroshot behavior.** Across the ten sampled base-model repairs, four make every zero-reference-count root collectible in the default non-TTL path, two introduce access-time cleanup heuristics, one unsubscribes records reachable from a released root, two add cleanup on invalidation or sweep paths, and one collects a zero-reference-count root only after it leaves the release buffer. None computes both the references needed to preserve cache-retained records and the references reachable from actively retained roots, then unsubscribes only the inactive difference. **Knowledge made available by repeated trials.** The visible leak initially suggests that unreferenced data should be collected more aggressively, but Trial 1 shows why that interpretation is incomplete: Relay's release buffer may deliberately keep a record after its last active consumer disappears. The record can remain valid as cached data even though its live resolver should stop consuming resources. Comparing the rejected and accepted repairs identifies two reference sets used during garbage collection: all references that keep records cached, and active references that determine which live subscriptions remain running.

Existing Relay code connects React effect cleanup to disposal of the operation retain. The ICL reference receives the original task issue and repository state together with the two-trial expe-

rience. One selected accepted repair modifies the downstream garbage-collection stage: when `RelayModernStore._collect()` runs, it preserves records reachable from cache-retained roots, computes the subset reachable from actively retained roots, and invokes the unsubscribe function for live state outside that active subset. All ten candidates sampled from the ICL reference are accepted under the pass@10 budget.

### C.5.3 Teacher-Generated Distillation Targets

Conditioned on the task-specific accumulated experience, the teacher explicitly rejects the all-or-nothing ownership model from Trial 1 and adopts the two-reference-set structure reached in Trial 2:

**Teacher Target: Separate Cached Records from Active Live Subscriptions**

**Relation to accepted repair. Accepted repair component: active-subscription cleanup during garbage collection.**

**Connection to accumulated trials.** Trial 1 tied subscription cleanup to deleting records with no active retainers, which broke Relay's release-buffer behavior. The accepted Trial 2 patch instead tracks all cache-retained references separately from references reachable through active consumers. The teacher selects this two-set distinction from the accumulated trials.

**Teacher-generated target.** "The correct approach is the two-set approach from Trial 2 that keeps the existing deletion behavior but adds an extra step to unsubscribe inactive subscriptions even when we keep the cached record."

The target assigns record deletion to all retained references and live work to active references. This resolves Trial 1's conflict: stopping the interval no longer requires discarding a cache-retained record.

### C.5.4 EPD Evaluation without Experience Context

**Observed behavior in the selected EPD patch.** At evaluation, the EPD-trained model receives the original task issue and repository state, but not the two-trial experience context. The selected lines show the garbage-collection stage of the repair; `maybeResolverSubscription` is the stored unsubscribe callback. Unchanged lines are omitted.

**EPD Evaluation Patch Excerpt: Live-Subscription Cleanup in `RelayModernStore._collect`**

```
const allReferences = new Set<DataID>();
const references = new Set<DataID>();
[...]
RelayReferenceMarker.mark(
 this._recordSource, selector, allReferences, [...]
);
[...]
if (refCount > 0) {
 RelayReferenceMarker.mark(
 this._recordSource, selector, references, [...]
 );
}
[...]
if (!references.has(dataID)) {
 const maybeResolverSubscription = RelayModernRecord.getValue(
 record,
 RELAY_RESOLVER_LIVE_STATE_SUBSCRIPTION_KEY,
 );
 if (maybeResolverSubscription != null) {
 maybeResolverSubscription();
 RelayModernRecord.setValue(record,
 RELAY_RESOLVER_LIVE_STATE_SUBSCRIPTION_KEY, null);
 }
}
[...]
if (!allReferences.has(dataID)) {
 this._recordSource.remove(dataID);
}
```

**Case-level outcome.** The selected lines show that, when garbage collection runs, records outside the active set are unsubscribed and their callbacks cleared, whereas deletion uses the broader cache-retained set. The external verifier exercises retain disposal followed by explicit garbage collection

rather than mounting and unmounting a React component. The displayed patch therefore establishes the store-cleanup stage, not the complete React unmount call chain. The candidate passes five `LiveResolvers` and 162 `RelayModernStore` tests, including release-buffer tests, and is accepted by the external verifier. Nine of ten EPD and all ten ICL candidates are accepted; zeroshot, SFT, and PPO each yield zero of ten.

### C.6 Case Study 4 on Curated SWE Task: Oversized Session Cookies in the Edge Runtime

#### C.6.1 Task and Repair Objective

**Task Issue Summary: Auth0 Edge-Runtime Session Cookies**

**Project and language.** `@auth0/nextjs-auth0` is a TypeScript authentication SDK for Next.js. The report concerns social login under the Next.js Edge Runtime.
**User-visible failure.** A large social-login profile causes the session cookie to be split into two pieces. In the Next.js Edge Runtime, the oversized session is not preserved correctly, so the user cannot complete login with a usable session and later access-token retrieval fails.
**Requested outcome.** Provide an Auth0-side repair that makes the split session cookie usable in the Edge Runtime.

#### C.6.2 Accumulated Experience and Experience-Conditioned ICL

**Repair attempts across two trials.** The first trial concentrates on response emission. Its rejection leaves the edit and all preceding inspection in context, allowing the second trial to examine the reverse direction of the same cookie protocol.

| Trial | Principal new change or revised hypothesis | Outcome |
|---|---|---|
| 1 | Replace the outgoing `Set-Cookie` value by appending individual cookie chunks. The response-writing path changes, but existing combined headers are still reconstructed with the original comma split. | Rejected |
| 2 | Repair both directions: reconstruct cookie boundaries without splitting attribute commas, then delete the combined header and append every reconstructed cookie separately. | Accepted |

Table 19: Evolution of the Edge Runtime cookie repair. The second trial broadens the repair from header emission to the complete read–modify–write protocol.

**Task-Specific Knowledge Made Available by Repeated-Trial Experience**

**Issue-and-repository-only zeroshot behavior.** All ten sampled base-model repairs change cookie emission. Four also change the read path: one uses `getSetCookie` when available, while three attempt to avoid splitting commas inside cookie attributes; a separate emission-focused patch mentions expiry-date commas but retains the original unsafe split. One of the three parsing changes uses a positive-boundary regular expression, but none implements the accepted stateful read–modify–write algorithm that reconstructs cookie boundaries, preserves attribute continuations, and emits each cookie separately. **Knowledge made available by repeated trials.** Trial 1 establishes that correcting only response emission is insufficient. Repository inspection after that failure exposes that the SDK receives several `Set-Cookie` values as one string. A comma can separate cookies, but it can also occur inside an attribute such as `Expires=Wed, 21 Oct ...`. The repair must infer a cookie boundary from the start of a new name/value pair rather than from every comma, preserve attribute continuations, and emit reconstructed cookies as separate response-header values.

The ICL reference receives the complete two-trial experience. One selected accepted repair treats header parsing and header emission as two directions of one serialization protocol: it reconstructs individual cookies from a combined value, preserves commas inside attributes, and appends each cookie separately. All ten candidates sampled from the ICL reference are accepted under the pass@10 budget.

### C.6.3 TEACHER-GENERATED DISTILLATION TARGETS

Conditioned on the task-specific accumulated experience, the teacher separates the two conclusions that distinguish Trial 2 from the rejected emission-only repair.

**Teacher Target 1: Extend the Repair to the Cookie Read Path**

**Relation to accepted repair. Accepted Trial 2 diagnosis: response emission alone is insufficient.**

**Connection to accumulated trials.** Trial 1 changes how reconstructed cookies are emitted but leaves the combined-header parser unchanged. The first teacher decision explicitly identifies that omitted read path.

**Teacher-generated target.** "Trial 1 failed because it only fixed the writing path, not the reading path."

**Teacher Target 2: Distinguish Cookie Boundaries from Attribute Continuations**

**Relation to accepted repair. Accepted stateful parsing behavior.**

**Connection to accumulated trials.** Trial 2 treats a post-comma fragment as a new cookie only when it begins with a cookie name/value pair; otherwise, the fragment remains part of the current cookie attribute.

**Teacher-generated target.** "only treats a part as a new cookie if it starts with `[a-z][a-z0-9-_]*=`"
[...]
"Correctly preserves commas inside existing cookie attributes"

Together, the first target identifies the read path omitted in Trial 1, while the second generates the stateful boundary rule implemented in Trial 2. Separate response headers cannot preserve chunked sessions if the read path has already divided an `Expires` attribute at its comma.

### C.6.4 EPD EVALUATION WITHOUT EXPERIENCE CONTEXT

**Observed behavior in the selected EPD patch.** At evaluation, the model trained with Experience Distillation receives the original task issue and repository state but not the two-trial experience context. The selected lines below implement both protocol directions; unchanged code connecting the parser result to the writer is omitted.

**EPD Evaluation Patch Excerpt: Reconstruct and Emit Chunked Session Cookies**

```
if (!value) return [];
const cookieList = [];
let currentCookie = '';
for (const part of value.split(',')) {
 if (!currentCookie) {
 currentCookie = part.trim(); continue;
 }
 const next = part.trimStart();
 if (/^[a-z][a-z0-9-_]*=/i.test(next)) {
 cookieList.push(currentCookie.trim());
 currentCookie = next;
 } else currentCookie += ',' + part;
}
if (currentCookie) cookieList.push(currentCookie.trim());
return cookieList;
[...]
res.headers.delete('set-cookie');
cookies.forEach(cookie => {
 res.headers.append('set-cookie', cookie);
});
```

**Case-level outcome.** The selected parser uses a name/value-prefix heuristic to distinguish a new cookie from an attribute continuation, rejoins the latter, and returns the reconstructed list. The displayed write path clears the combined header and appends each reconstructed cookie. This is the concrete accepted implementation, not a claim that the regular expression defines every valid cookie-name boundary. The patch therefore exhibits both protocol directions observed under experience-conditioned ICL without receiving the two-trial experience context at evaluation. All 49 tests in the candidate's reported utility-test run pass. Among the nine EPD candidates retained for this

task, eight are accepted; all ten ICL candidates are accepted, whereas zeroshot, SFT, and PPO each produce zero accepted candidates from ten samples.

### C.7 Case Study 5 on Curated SWE Task: Configurable GraphQL Root Type Names

#### C.7.1 Task and Repair Objective

**Task Issue Summary: GraphQL Mesh Root Type Names**

**Project and language.** GraphQL Mesh composes schemas and applies TypeScript transforms. Its encapsulation transform wraps root operations from one source under a generated field.
**User-visible failure.** The transform assumes that every source schema names its root operation types `Query`, `Mutation`, and `Subscription`. A schema using names such as `QueryRoot` cannot be encapsulated because the transform looks up and rewrites the wrong outer type.
**Requested behavior.** Allow users to configure the query, mutation, and subscription outer-root names used by the encapsulation transform.

#### C.7.2 Accumulated Experience and Experience-Conditioned ICL

**Repair attempts across six trials.** The issue appears to require three parameter substitutions, but the accumulated trials show that the names participate in schema lookup, generated wrapper construction, lifecycle-dependent defaults, and public configuration.

| Trials | Principal new change or revised hypothesis | Outcome |
|---|---|---|
| 1–3 | Add the three public configuration fields and generated artifacts, generalize source-root lookup to string keys, and preserve stable generated names. These attempts still resolve absent options to conventional names before the source schema is available. | Rejected |
| 4 | Also derive the generated inner wrapper names from the configured outer names. This couples two names that serve different roles in the transformed schema. | Rejected |
| 5 | Move root resolution to schema time and detect defaults from the input schema, but still derive generated wrapper names from the resolved outer names. | Rejected |
| 6 | Restore stable generated names such as `SomeSourceQuery` while retaining the configuration and schema-time source-root resolution introduced earlier. | Accepted |

Table 20: Evolution of the GraphQL Mesh repair. The repeated failures separate source-schema lookup names from names created by the encapsulation transform.

**Task-Specific Knowledge Made Available by Repeated-Trial Experience**

**Issue-and-repository-only zeroshot behavior.** All ten sampled trajectories refer to the three explicit root-name overrides named in the issue, but only five modify a public configuration artifact and only three integrate the options into runtime behavior. None defers default resolution until the input schema is available. Of the three runtime patches, two preserve stable generated names such as `SomeSourceQuery`, whereas one derives generated names from configured source-root names. The complete separation between schema-discovered source roots and stable generated wrapper names is therefore not reliably composed under issue-and-repository-only evaluation. **Knowledge made available by repeated trials.** Trial 5 defers source-root lookup until the schema is available, but incorrectly uses the detected source name as the generated wrapper name. Trial 6 preserves the schema-time lookup while restoring stable generated names such as `SomeSourceQuery`. For example, if the source named `SomeSource` exposes a query root called `QueryRoot`, the transform must find the existing `QueryRoot` type but still create the wrapper name `SomeSourceQuery`. The accepted repair therefore separates names discovered from the source schema from wrapper names generated by GraphQL Mesh and exposes the explicit overrides through the public configuration surfaces.

The ICL reference receives all six trials, and nine of ten sampled candidates are accepted. In one selected accepted patch, each source-root name is resolved from explicit configuration, then from the available schema, and finally from the conventional fallback. The generated wrapper name remains based on the source name. The query branch below shows this separation; the same patch applies the corresponding structure to mutation and subscription and adds all three options to the public configuration interfaces.

**Selected Accepted ICL Patch Excerpt: Separate Source and Generated Root Names**

```
if (this.applyTo.query) {
 const outerName = this.queryOuterType ||
 originalWrappingSchema.getQueryType()?.name || 'Query';
 const transform = new WrapType(
 outerName, `${this.name}Query`, this.name);
 [...]
}
```

### C.7.3 Teacher-Generated Distillation Targets

Conditioned on the accumulated experience, separate decisions in the selected branch-packed sequence express schema-time root resolution, stable generated naming, and the corresponding public configuration.

**Teacher Target 1: Resolve Defaults after the Source Schema Is Available**

**Relation to accepted repair. Schema-time fallback introduced in Trial 5.**

**Connection to accumulated trials.** The first three trials resolve absent overrides before the input schema is available. Trial 5 moves this decision to schema transformation, where GraphQL Mesh can inspect the actual source-root names.

**Teacher-generated target.** "Move `WrapType` creation from constructor to `generateSchemaTransforms` so we can get the actual root names from the original schema if custom names are not provided"

**Teacher Target 2: Preserve Stable Generated Wrapper Names**

**Relation to accepted repair. Decisive Trial 6 correction.**

**Connection to accumulated trials.** Trial 5 obtains the correct source-root names but also uses them to derive generated wrapper names. Trial 6 separates those roles and restores the original Mesh-generated naming pattern.

**Teacher-generated target.** "Keep the original inner naming pattern to maintain backwards compatibility"

**Teacher Target 3: Expose Root-Name Overrides through Public Configuration**

**Relation to accepted repair. Public configuration carried through the accepted repair.**

**Connection to accumulated trials.** Trials 1–3 add the three override fields. The selected teacher sequence regenerates those fields in the YAML schema and carries them into the generated JSON schema and TypeScript interface.

**Teacher-generated target.** "First, add the new configuration options to `yaml-config.graphql` as requested."
[...]
"Updated `config-schema.json` with new configuration [...] Updated `config.ts` TypeScript types with new optional properties"

These decisions occur at different branch points in the selected sequence. Collectively, they express the distinction established by Trials 5–6: inspect the available source schema to find existing root names, preserve GraphQL Mesh's generated naming convention, and expose explicit overrides through public configuration.

### C.7.4 EPD Evaluation without Experience Context

**Observed behavior in the selected EPD patch.** At evaluation, the EPD-trained model receives the original task issue and repository state, but not the six-trial experience context. The runtime excerpt implements the accepted distinction completed in Trial 6: configuration or the available schema determines which existing root type is wrapped, while GraphQL Mesh continues to generate stable names from the source name.

**EPD Evaluation Patch Excerpt: Root Resolution in `encapsulate/src/index.ts`**

*Constructor configuration:*

```
this.customQueryOuterType = config?.queryOuterType;
this.customMutationOuterType = config?.mutationOuterType;
this.customSubscriptionOuterType = config?.subscriptionOuterType;
[...]
```

*Query branch in `generateSchemaTransforms`:*

```
if (this.applyTo.query) {
 const queryOuterType = this.customQueryOuterType ||
 originalWrappingSchema.getQueryType()?.name || 'Query';
 const transformedName = `${this.name}Query`;
 const transform = new WrapType(
 queryOuterType, transformedName, this.name);
 [...]
}
[...]
```

*Mutation branch in `generateSchemaTransforms`:*

```
if (this.applyTo.mutation) {
 const mutationOuterType = this.customMutationOuterType ||
 originalWrappingSchema.getMutationType()?.name || 'Mutation';
 const transformedName = `${this.name}Mutation`;
 const transform = new WrapType(
 mutationOuterType, transformedName, this.name);
 [...]
}
[...]
```

*Subscription branch in `generateSchemaTransforms`:*

```
if (this.applyTo.subscription) {
 const subscriptionOuterType = this.customSubscriptionOuterType ||
 originalWrappingSchema.getSubscriptionType()?.name || 'Subscription';
 const transformedName = `${this.name}Subscription`;
 const transform = new WrapType(
 subscriptionOuterType, transformedName, this.name);
 [...]
}
```

The public-configuration excerpt shows the three override fields first introduced in Trials 1–3. The selected diff adds the same options to the transform's YAML schema, the TypeScript configuration interface, and the generated JSON schema.

**EPD Evaluation Patch Excerpt: Root-Name Options in Public Configuration**

*yaml-config.graphql:*

```
queryOuterType:  String
mutationOuterType:  String
subscriptionOuterType:  String
[...]
```

*packages/types/src/config.ts:*

```
queryOuterType?:  string;
mutationOuterType?:  string;
subscriptionOuterType?:  string;
[...]
```

*packages/types/src/config-schema.json:*

```
"queryOuterType":  { "type":  "string" }
[analogous mutation and subscription fields]
```

**Case-level outcome.** The selected EPD patch exhibits the outer-versus-generated naming distinction identified through experience-conditioned ICL for query, mutation, and subscription roots. It also exposes the corresponding public configuration fields. Its tests report 146 passed and one skipped. Nine of ten EPD candidates and nine of ten ICL candidates are accepted, compared with zero of ten candidates from each of zeroshot, SFT, and PPO.

## C.8 Case Study 6 on Curated SWE Task: Squash-Merge Commit Message Selection

### C.8.1 Task and Repair Objective

**Task Issue Summary: GitHub CLI Squash-Merge Messages**

**Project and language.** GitHub CLI is a Go command-line client. The task concerns the behavior of `gh pr merge` for squash merges.
**User-visible failure.** For squash merges, the existing command derives the resulting commit message from the last commit, which may be an uninformative message such as a lint or rebase update. The issue requests a choice among the pull-request title, the last commit message, and a custom message.
**Requested behavior.** Preview the squash-merge commit message in the CLI and let the user choose the pull-request title, the last commit message, or a custom message.

### C.8.2 Accumulated Experience and Experience-Conditioned ICL

**Repair attempts across five trials.** The requested choice appears local to a terminal prompt, but each attempt exposes another part of the command-to-API path.

| Trial | Principal new change or revised hypothesis | Outcome |
|---|---|---|
| 1 | Add a three-way squash-message prompt and extend the merge mutation to accept a selected headline and body. Prompting enters command paths that previously completed without input. | Rejected |
| 2 | Restrict prompting to fewer flows so existing command tests complete, but leave the interaction condition and headless default unresolved. | Rejected |
| 3 | Prompt whenever a squash merge runs on a TTY, including explicit `--squash`. This still treats terminal availability as equivalent to interactive intent. | Rejected |
| 4 | Establish pull-request title/body as the deterministic squash default, but still gate the prompt on TTY alone. | Rejected |
| 5 | Require both interactive mode and stdin TTY before prompting, while retaining the query data, mutation fields, and headless default introduced earlier. | Accepted |

Table 21: Evolution of the GitHub CLI repair. The trials expand a terminal prompt into an explicit contract for interactive and headless command execution.

**Task-Specific Knowledge Made Available by Repeated-Trial Experience**

**Issue-and-repository-only zeroshot behavior.** All ten sampled trajectories locate the merge command and trace its call into `api.PullRequestMerge`, but every trajectory terminates during repository inspection without applying an edit. Consequently, none implements the query extension, pull-request-title/body default, interaction gate, three-way selection, or mutation propagation that appear together in the accepted repair. Under issue-and-repository-only evaluation, the complete command-to-API behavior is therefore not reliably composed. **Knowledge made available by repeated trials.** The repeated trials establish the title/body default before prompting, restrict the prompt to interactive mode when stdin is attached to a TTY, fetch the last commit for the corresponding option, and pass the selected fields to the merge mutation.

The ICL reference receives all five trials. One selected accepted repair separates interactive and headless behavior, obtains the data needed for every offered option, and carries the selected message through the GraphQL mutation. All ten ICL candidates are accepted under the pass@10 budget.

### C.8.3 Teacher-Generated Distillation Targets

Conditioned on the accumulated experience, the selected teacher targets retain Trial 1's query and API path, Trial 4's deterministic headless default, and Trial 5's corrected interaction gate.

**Teacher Target 1: Use the Pull-Request Message as the Headless Default**

**Relation to accepted repair. Accepted Trial 4 component: deterministic noninteractive behavior.**

**Connection to accumulated trials.** Trials 1–3 expose command paths that must not block for input. Trial 4 establishes the pull-request title and body as the squash-message default, which Trial 5 retains.

**Teacher-generated target.** "set the default commit message for squash merges to be the PR title + PR body (fixing the core issue)"

**Teacher Target 2: Gate the Three-Way Prompt on Interactive Mode and TTY**

**Relation to accepted repair. Decisive Trial 5 correction.**

**Connection to accumulated trials.** Trial 1 introduces the three choices, but Trials 2–4 do not yet distinguish terminal availability from interactive intent. Trial 5 requires both interactive mode and stdin attached to a TTY before presenting the prompt.

**Teacher-generated target.** "Add the interactive prompt when in interactive mode and TTY to allow users to select between the three options: PR title, last commit, custom message"

**Teacher Target 3: Fetch and Propagate the Selected Message**

**Relation to accepted repair. Accepted Trial 1 data-and-API component retained through Trial 5.**

**Connection to accumulated trials.** Trial 1 extends the pull-request query and merge mutation so that the three-way choice can read the last commit and transmit the selected headline and body. Trials 2–5 revise when prompting occurs and what the default should be, while retaining this cross-layer path.

**Teacher-generated target.** "Update all GraphQL queries that fetch commits to get the last commit's message"
[...]
"Update PullRequestMerge function to accept commitHeadline and commitBody parameters"
[...]
"Modify the mutation to add CommitHeadline and CommitBody when they're not empty"
[...]
"Update all the calls to api.PullRequestMerge to include the new parameters"

Together, the three targets cover the deterministic default, the interactive override, and the query-to-mutation data path combined by the accepted trial.

### C.8.4 EPD Evaluation without Experience Context

**Observed behavior in the selected EPD patch.** At evaluation, the EPD-trained model receives the original task issue and repository state, but not the five-trial experience context. The first excerpt adds a `Message` field to each queried commit node and requests it from the last commit.

**EPD Evaluation Patch Excerpt: Fetch the Last Commit Message**

```
Message string
[...]
commits(last:  1) {
 nodes {
 commit { message }
 }
}
```

The next excerpt shows how the accepted repair combines Trial 4's deterministic headless default with Trial 5's interaction gate. The two nondefault choices replace the headline and body with the last commit or a custom message.

**EPD Evaluation Patch Excerpt: Select the Squash-Merge Message**

```
prTitle := fmt.Sprintf("%s (#%d)", pr.Title, pr.Number)
if mergeMethod == api.PullRequestMergeMethodSquash {
 commitHeadline = prTitle; commitBody = pr.Body
}
[...]
if mergeMethod == api.PullRequestMergeMethodSquash &&
 opts.InteractiveMode && opts.IO.IsStdinTTY() {
```

**EPD Evaluation Patch Excerpt: Select the Squash-Merge Message (continued)**

```
 lastCommitMessage = strings.TrimSpace(
 pr.Commits.Nodes[len(pr.Commits.Nodes)-1].Commit.Message)
 options := []string{
 fmt.Sprintf("Use pull request title:  %q", prTitle),
 fmt.Sprintf("Use last commit message:  %q", lastCommitMessage),
 "Enter custom commit message",
 }
 [...]
 switch(choice) {
 case 1:
 if lines := strings.SplitN(lastCommitMessage,
 "\n", 2); len(lines) >= 2 {
 commitHeadline = lines[0]
 commitBody = strings.TrimSpace(lines[1])
 } else { [...] }
 case 2:
 [...]  SurveyAskOne(editorQuestion, &customMessage)
 if lines := strings.SplitN(strings.TrimSpace(customMessage),
 "\n", 2); len(lines) >= 2 {
 commitHeadline = lines[0]
 commitBody = strings.TrimSpace(lines[1])
 } else { [...] }
 }
}
```

The final excerpt shows that the selected values cross the GraphQL API boundary rather than remaining local command state.

**EPD Evaluation Patch Excerpt: Pass the Message to the Merge Mutation**

```
if commitHeadline != "" {
 commitHeadline := githubv4.String(commitHeadline)
 input.CommitHeadline = &commitHeadline
}
if commitBody != "" {
 commitBody := githubv4.String(commitBody)
 input.CommitBody = &commitBody
}
```

**Case-level outcome.** The selected EPD patch implements the headless default, interactive override, and query-to-mutation path identified across the five trials. It exhibits this command/API contract without receiving the experience context at evaluation. The squash, non-TTY, and interactive merge-command tests pass. Nine of ten EPD candidates are accepted, compared with ten of ten ICL candidates and zero of ten candidates from each of zeroshot, SFT, and PPO.

### C.9 Case Study 7 on Curated SWE Task: Stale Duplicate Error after Circular-Reference Recovery

#### C.9.1 Task and Repair Objective

**Task Issue Summary: Gauge Circular-Reference Recovery**

**Project and language.** Gauge is a Go test-automation framework. A `.cpt` file defines reusable *concepts*, each expanding to a sequence of test steps.
**User-visible failure.** After a user creates and then removes a circular reference in a `.cpt` file, Gauge continues to report `Duplicate concept definition found`; the report provides no additional logs.
**Expected behavior.** Removing the circular reference should also remove the stale duplicate-definition error.

#### C.9.2 Accumulated Experience and Experience-Conditioned ICL

**Repair attempts across two trials.** Trial 1 modifies both the editor caller and parser cleanup. Trial 2 retains those edits and moves file deduplication to the shared discovery utilities used outside the editor.

| Trial | Principal new change or revised hypothesis | Outcome |
|---|---|---|
| 1 | Deduplicate concept-file inputs in the language-server caller and replace direct `ConceptsMap` deletion with `conceptDictionary.Remove`, which clears both parser maps. Shared non-editor callers can still receive duplicate files. | Rejected |
| 2 | Retain dictionary-level cleanup and move concept/specification-file deduplication into the shared discovery utilities used by all callers. | Accepted |

Table 22: Evolution of the Gauge repair. Trial 2 broadens caller-local deduplication into a shared file-discovery invariant while retaining complete parser-state cleanup.

**Task-Specific Knowledge Made Available by Repeated-Trial Experience**

**Issue-and-repository-only zeroshot behavior.** All ten sampled trajectories terminate during parser and file-discovery inspection without applying an edit. One explicitly diagnoses that circular-reference cleanup should call `conceptDictionary.Remove` rather than delete only from `ConceptsMap`, but none implements that change; none adds either caller-local or shared file deduplication. The complete combination of two-map cleanup and shared deduplication is therefore not reliably produced under issue-and-repository-only evaluation.
**Knowledge made available by repeated trials.** Trial 1 combines caller-local deduplication with complete dictionary removal, but remains rejected. Trial 2 retains the parser cleanup and moves deduplication into shared `GetConceptFiles` and `GetSpecFiles`. Dictionary-level removal prevents a failed concept from remaining in the secondary `constructionMap`, while shared deduplication prevents the same concept file from re-entering the parser through repeated discovery paths. The accepted state therefore removes failed concepts from every internal representation and makes discovery idempotent across callers while leaving genuine duplicate definitions reportable.

The ICL reference receives both trials. One selected accepted repair uses the dictionary's complete removal operation and deduplicates files before parsing, so the same behavior applies to CLI and editor paths. All ten ICL candidates are accepted under the pass@10 budget.

### C.9.3 Teacher-Generated Distillation Targets

Conditioned on the accumulated experience, the teacher identifies shared file discovery as the missing scope change and separately retains dictionary-level cleanup. Its first quoted diagnosis overlooks the dictionary cleanup already added in Trial 1, but the generated edit matches the accepted Trial 2 change from caller-local to shared deduplication.

**Teacher Target 1: Move Deduplication into Shared File Discovery**

**Relation to accepted repair. Accepted Trial 2 scope correction; the diagnosis below partially misstates Trial 1.**

**Connection to accumulated trials.** Trial 1 already changes parser cleanup and deduplicates in the language-server caller. Trial 2 applies `uniqueStrings` in the shared `GetConceptFiles` and `GetSpecFiles` utilities.

**Teacher-generated target.** "So my previous fix in trial 1 only fixed diagnostics (LSP/editor), but didn't fix the core GetConceptFiles. The problem is that the bug can also happen on CLI when running tests, not just in the editor."
[...]
"Added deduplication to both `GetSpecFiles` and `GetConceptFiles` using a new helper function `uniqueStrings`."

**Teacher Target 2: Remove Circular-Reference State from Both Parser Maps**

**Relation to accepted repair. Accepted repair component retained from Trial 1.**

**Connection to accumulated trials.** Trial 1 already replaces direct deletion from `ConceptsMap` with `conceptDictionary.Remove`; Trial 2 retains this change while correcting deduplication scope. The quote below accurately states the two-map cleanup performed by that operation.

**Teacher-generated target.**
"Changed the circular reference cleanup from just deleting from `ConceptsMap` to using `conceptDictionary.Remove()` which correctly cleans up both `ConceptsMap` and `constructionMap`."

Together, the two targets specify the two accepted repair components. The EPD patch below omits Trial 1's now-redundant language-server-local deduplication.

#### C.9.4 EPD Evaluation without Experience Context

**Observed behavior in the selected EPD patch.** At evaluation, the EPD-trained model receives the original task issue and repository state, but not the two-trial experience context. The selected patch contains both shared operations identified above:

**EPD Evaluation Patch Excerpt: Shared Duplicate-Concept Recovery**

```
ValidateConcepts:
conceptDictionary.Remove(concept.ConceptStep.Value)
[...]
GetSpecFiles:
return uniqueStrings(specFiles)
[...]
GetConceptFiles:
return uniqueStrings(files)
```

**Case-level outcome.** The selected EPD patch removes failed concepts through the dictionary abstraction, which updates both parser maps, and applies `uniqueStrings` to both discovery paths. The patch exhibits the parser-state and input-idempotence behavior observed under experience-conditioned ICL without receiving the two-trial experience context at evaluation. The parser reports 198 passing tests; the utility and Gauge suites also pass, including checks that intentional duplicates remain detected. Nine of ten EPD candidates are accepted, compared with ten of ten ICL candidates and zero of ten candidates from each of zeroshot, SFT, and PPO.

### C.10 Case Study 8 on Curated SWE Task: Backend-Relative Stored Media URLs in Strapi

#### C.10.1 Task and Repair Objective

**Task Issue Summary: Strapi Backend-Relative Media URLs**

**Project and language.** Strapi is a JavaScript/TypeScript content platform. The report concerns its media-library administration interface.
**User-visible failure.** Opening the media library produces a white page with `TypeError: Failed to construct 'URL': Invalid URL`. The reporter notes that the stored `files.url` values all begin with a slash and identifies this data characteristic as the apparent trigger.
**Expected behavior.** The media-library page should open when existing records contain slash-prefixed media URLs.

#### C.10.2 Accumulated Experience and Experience-Conditioned ICL

**Repair attempts across four trials.** The repeated trials explore stored-media parsing, add defensive provider handling, and eventually identify Strapi's backend-prefix helper as the repository-specific mechanism for stored paths. The accepted fourth trial still broadens the add-from-URL schema; the teacher later restores strict validation for user-entered URLs.

| Trial | Principal new change or revised hypothesis | Outcome |
|---|---|---|
| 1 | Supply a browser base to frontend URL parsers, including the "add from URL" schema. Relative stored paths become parseable, but user-entered relative URLs also become valid. | Rejected |
| 2 | Retain the frontend changes and add defensive handling in the storage-provider URL utilities. The repair still applies one policy to distinct data sources. | Rejected |
| 3 | Add guarded frontend fallbacks while keeping the user-input schema permissive toward relative values. | Rejected |
| 4 | Replace ad hoc bases with Strapi's backend-prefix helper for stored paths and retain defensive provider handling. The patch fixes the reported failure but still applies the prefix helper inside the user-input schema. | Accepted |

Table 23: Evolution of the Strapi repair. The trials establish stored-path normalization but leave an overbroad user-input schema change for the teacher to correct.

**Task-Specific Knowledge Made Available by Repeated-Trial Experience**

**Issue-and-repository-only zeroshot behavior.** All ten sampled trajectories locate the one-argument `new URL(...)` sites relevant to slash-prefixed paths, but five terminate before applying an edit. The other five add a browser-derived base in `appendSearchParamsToUrl`; four also change filename extraction, three also relax the user-input schema, and one adds local fallbacks. None uses Strapi's `prefixFileUrlWithBackendUrl` helper or edits the provider utilities. Thus none combines the repository's backend-prefix helper in the reported media-library path with defensive provider handling; three edited candidates also broaden user-input validation.
**Knowledge made available by repeated trials.** Trial 4 replaces browser-origin resolution with the repository-specific backend-prefix helper across the affected stored-media utilities and retains defensive provider handling. The same accepted trial still relaxes the "add from URL" schema, so preserving strict validation for user-entered URLs is a later teacher refinement rather than knowledge established by the repeated-trial outcome.

The ICL reference receives all four trials. One selected accepted repair applies the existing prefix helper to stored media paths and adds defensive provider handling; like Trial 4, it also relaxes the user-input schema. All ten ICL candidates are accepted under the pass@10 budget. The teacher sequence retains the trial-derived stored-path and provider changes and independently restores the original user-input policy.

### C.10.3 Teacher-Generated Distillation Targets

Conditioned on the task-specific accumulated experience, the teacher separates the stored-path normalization reached in Trial 4 from the schema policy that Trial 4 changed unnecessarily.

**Teacher Target 1: Apply the Backend Prefix before URL Parsing**

**Relation to accepted repair. Accepted stored-path repair component.**

**Connection to accumulated trials.** Trials 1–3 add browser-derived bases or local fallbacks. Trial 4 identifies Strapi's existing backend-prefix helper; the teacher explicitly reuses that repository abstraction before parsing a potentially relative path.

**Teacher-generated target.** "the function is called with a URL that might still be relative [...] every other function in the codebase that needs an absolute file URL handles it by calling `prefixFileUrlWithBackendUrl`"

**Teacher Target 2: Restore Absolute-URL Validation for User Input**

**Relation to accepted repair. Teacher revision of Trial 4's accepted patch.**

**Connection to accumulated trials.** Trial 4 also invokes the backend-prefix helper from the "add from URL" schema, making relative user input valid. The teacher distinguishes that form from stored media paths and restores the schema's original absolute-URL policy.

**Teacher-generated target.** "`urlYupSchema.js` is already back to its original state, which is correct. We shouldn't modify it at all because it's meant to validate that user-input URLs are already absolute URLs."

**Teacher Target 3: Guard Provider-Side URL Parsing**

**Relation to accepted repair. Defensive behavior retained from the repeated trials.**

**Connection to accumulated trials.** Trials 2–4 add provider-side handling so malformed or relative values cannot produce an unhandled server-side parse failure. The selected teacher sequence explicitly revisits this provider path.

**Teacher-generated target.** "Now let me also add the defensive error handling to the AWS S3 provider utils"

Together, the three targets separate backend-relative stored paths from user-entered external URLs and retain defensive provider behavior: normalize stored paths with the repository's backend-prefix helper, preserve absolute-only validation for user input, and guard provider-side parsing.

### C.10.4 EPD Evaluation without Experience Context

**Observed behavior in the selected EPD patch.** At evaluation, the model trained with Experience Distillation receives the original task issue and repository state but not the four-trial experience context. The complete selected patch leaves `urlYupSchema.js` unchanged. The first excerpt shows the reported media-library fix in `appendSearchParamsToUrl` and defensive reuse of the same helper in filename extraction.

**EPD Evaluation Patch Excerpt: Prefix Stored Media Paths before Parsing**

```
appendSearchParamsToUrl:
const prefixedUrl =
 prefixFileUrlWithBackendUrl(url);
const urlObj = new URL(prefixedUrl);
[...]
getFilenameFromURL:
function getFilenameFromURL(url) {
 const prefixedUrl = prefixFileUrlWithBackendUrl(url);
 return new URL(prefixedUrl).pathname.split('/').pop();
}
```

The provider-side excerpt shows the complementary behavior: an invalid provider URL no longer produces an unhandled parse failure.

**EPD Evaluation Patch Excerpt: Guard Provider-Side URL Parsing**

```
try {
 url = new URL(fileUrl);
} catch (err) {
 return false;
}
```

**Case-level outcome.** The helper use in `appendSearchParamsToUrl` and the provider guard correspond to trial-derived behavior; leaving `urlYupSchema.js` unchanged corresponds to the teacher's later refinement. The patch also reuses the helper for filename extraction. Nine of ten EPD candidates are accepted without the four-trial experience context at evaluation, compared with ten of ten ICL candidates and zero of ten candidates from each of zeroshot, SFT, and PPO.

### C.11 Cross-Case Findings and Limitations

Across the eight cases, the same base model exhibits different repair behavior when conditioned on repeated-trial experience. The ten issue-and-repository-only samples for each case do not produce an accepted repair, whereas the experience-conditioned ICL samples implement the task-specific behavior summarized in Table 24. Selected teacher targets encode all or part of this behavior. A selected sequence can also revisit a rejected collection hypothesis, as in MapStore2, or refine an accepted collection patch, as in Strapi; the Billboard.js sequence does not regenerate every edit in the accepted working tree. The selected EPD patches implement the summarized behavior at evaluation without the experience context.

| **Case** | **Behavior across ten issue-and-repository-only zeroshot samples** | **Behavior available through experience-conditioned ICL** | **Behavior observed in selected EPD patch** |
|---|---|---|---|
| Geometry reset | Local filter and state edits; no complete reset-to-refresh path. | Emit an explicit null reset and preserve it through query refresh. | Emits `value: null` and preserves the geometry update in the query epic. |
| Tooltip alignment | Nearest-point lookup and local guards; no coordinated cross-layer semantics. | Combine representation-aware lookup with absence handling across propagation, API, and DOM layers. | Uses shared-x binary search, a multiple-x fallback, and explicit absence guards. |
| Subscription cleanup | Garbage-collection or unsubscribe edits that conflate record and subscription lifetimes. | Separate cache-retained records from active live subscriptions during store collection. | Tracks retained and active references separately and unsubscribes their difference when garbage collection runs. |
| Session cookies | Emission changes and partial comma handling; no complete parser–writer protocol. | Reconstruct cookie boundaries while preserving attribute commas, then emit separate headers. | Detects cookie boundaries, preserves `Expires` fragments, and appends separate headers. |
| GraphQL roots | All ten reference explicit overrides; five edit public configuration, three integrate runtime use, and none resolves absent overrides from the input schema. | Separate schema-discovered source roots from stable generated wrapper names. | Resolves source roots at schema time and preserves generated names across configuration surfaces. |
| Squash messages | All ten trace the merge command into the API helper but stop before editing; no query, default, prompt-gating, or mutation change is implemented. | Combine a deterministic headless default, gated interactive choices, and API propagation. | Supplies the default, gates prompts on interactive mode and TTY, and passes mutation fields. |
| Parser recovery | All ten stop before editing; one diagnoses two-map cleanup, and none implements caller-local or shared deduplication. | Combine complete parser cleanup with shared file-discovery deduplication. | Uses dictionary-level removal and deduplicates both shared discovery paths. |
| Media URLs | Five add browser-based handling in an admin URL utility, three also relax input validation, and none uses the backend-prefix helper or adds provider guards. | Normalize stored paths with the backend helper and retain provider-side guards. | Prefixes stored paths, adds defensive provider parsing, and preserves strict input validation after teacher refinement. |

Table 24: Behavioral comparison across eight cases. The zeroshot column audits all ten issue-and-repository-only samples; the ICL and EPD columns summarize accepted experience-conditioned repairs and one selected EPD patch generated without the experience context.

**Outcome contrast across the selected cases.** Under the pass@10 budget, experience-conditioned ICL solves all eight cases with 79 accepted candidates out of 80, whereas zeroshot solves none. EPD also solves all eight without the experience context, yielding 72 accepted candidates out of 79 valid samples; SFT and PPO yield no accepted candidates in 80 samples per method. These counts establish the case outcomes, while the selected teacher decisions and EPD patches show the task-specific behavior observed before and after distillation.

**Scope of the qualitative evidence.** Each teacher excerpt and EPD patch is one selected qualitative sample. Their correspondence documents an observed repair pattern but does not identify a unique causal path or estimate how frequently EPD uses that strategy. Because the cases are selected for outcome contrast, aggregate pass@1 in Table 1 remains the evidence for overall performance.