PREDICTING PRIVILEGED INFORMATION EFFECTIVENESS FOR ON-POLICY SELF-DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

On-policy self-distillation (**OPSD**) enables large language models (LLMs) to learn from privileged information (**PI**) without a separate teacher model. However, prior work largely treats PI as a fixed design choice, leaving the effects of its content and construction poorly understood. This challenge is particularly important in multi-turn agentic settings, where trajectories contain actions, observations, and reasoning that can be transformed into PI in many ways. We systematically evaluate 14 PI configurations across three multi-turn agentic benchmarks and three student models that vary in scale and reasoning mode. We find that PI effectiveness depends on its content, generation source, and the student model. Information-dense trajectories and PI produced by more capable generators generally yield larger gains, whereas thinking models benefit less consistently. To assess PI before training, we compare PI utility with two KL-based metrics: sampled-token reverse KL and error-step KL contrast. The KL-based metrics track post-training performance more consistently than PI utility, which can be distorted by information leakage. Finally, effective PI remains beneficial when combined with GRPO, allowing the combined objective to outperform a strong GRPO baseline. Overall, our results provide practical guidance for effective PI construction and evaluation within OPSD training.

1 INTRODUCTION

On-policy self-distillation (OPSD) enables LLMs to iteratively improve by turning successful solutions, environment feedback, and experience from their own interactions into training signals without a separate teacher model (Penaloza et al., 2026; Zhao et al., 2026; Hübotter et al., 2026; Shenfeld et al., 2026). Related self-distillation methods have already been used to train coding agents such as Composer 2.5 and agents for long-horizon professional tasks (Cursor Team, 2026; Kale & Trajectory Labs, 2026). In OPSD, the model learns from its own trajectories by matching the same model conditioned on PI, thereby internalizing information unavailable at inference time.

The effectiveness of OPSD, however, depends on what the teacher is allowed to see. In general, PI can take many forms, including ground-truth answers (Zhao et al., 2026), demonstrations from prior tasks (Shenfeld et al., 2026), environment feedback (Hübotter et al., 2026), actions from successful trajectories (Penaloza et al., 2026), or hints derived from gold solutions or trajectories, such as summaries (Penaloza et al., 2026), skills (Wang et al., 2026), checklists, structured criteria such as rubrics (Rezaei et al., 2026; Gu et al., 2026), and targeted feedback (Yeo et al., 2026; Cursor Team, 2026). These choices determine both the information available to the teacher and how that information changes its distribution relative to the student. However, especially on multi-turn agentic tasks, existing studies evaluate only a limited range of PI formats across a small set of models and tasks (Lu et al., 2026; Yang et al., 2026a; Wang et al., 2026; Yeo et al., 2026; Yang et al., 2026b). It remains unclear what makes PI produce useful learning signals for the student policy, whether the same choices work across agentic domains and model types, and, critically, whether PI quality can be assessed before training. Reliable offline metrics would allow candidate PI formats and sources to be compared before training, avoiding a separate costly OPSD run for every candidate and focusing compute on the most promising PI.

We study these questions through a systematic evaluation of OPSD on three multi-turn agentic benchmarks: ALFWorld (Shridhar et al., 2021), AppWorld (Trivedi et al., 2024), and τ^2-retail (Barres et al., 2025). Figure 1 summarizes the PI formats and offline metrics evaluated in our study. We

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

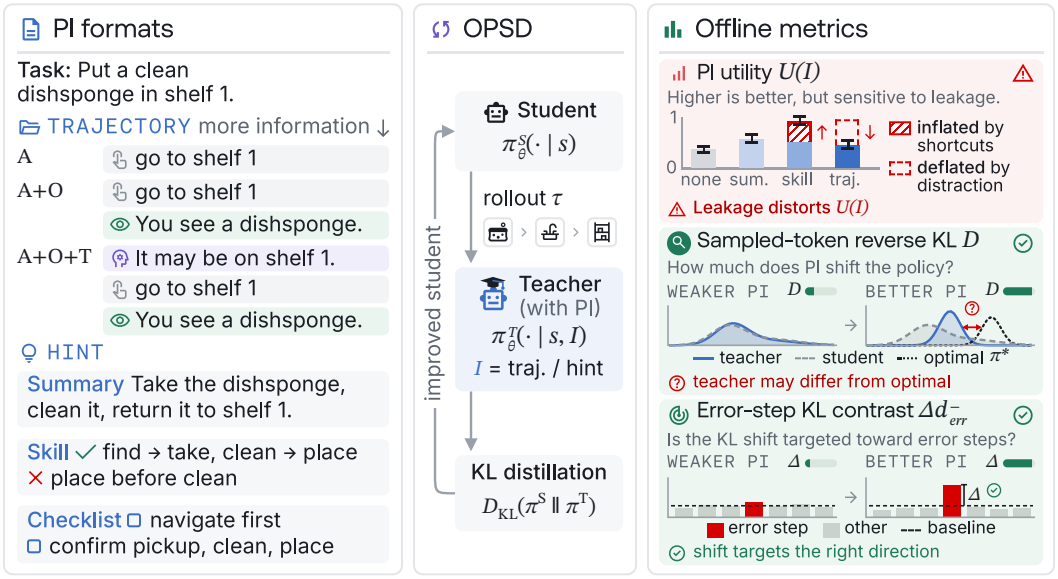


Figure 1: Overview of the PI formats and offline metrics studied. **Left—PI formats:** reference trajectories retain actions (**A**), add observations (**A+O**), or also include thoughts (**A+O+T**). Compressed hints include a plain **Summary**, a **Skill** with mistake analysis and reusable strategies, and a **Checklist** of task requirements derived from reference and failed student trajectories. **Right—offline metrics: PI utility** measures the teacher's reward gain from PI, but leakage can inflate or deflate it. **Sampled-token reverse KL** D measures shift magnitude; better PI is generally associated with a larger D, but D does not indicate whether the shift is beneficial. **Error-step KL contrast** $\Delta\bar{d}_{\mathrm{err}}$ measures whether that shift targets the identified errors.

cover 14 PI configurations spanning reference trajectories and self- and expert-generated hints, including summaries, skills, and checklists. We train Qwen3-4B-Instruct, Qwen3-4B-Thinking, and Qwen3-30B-A3B-Instruct (Yang et al., 2025), allowing us to compare PI effects across instruction and thinking models and across model scales. Beyond downstream performance, we evaluate three offline metrics: **PI utility** (Penaloza et al., 2026) compares teacher reward with and without PI, but can be distorted by information leakage; **sampled-token reverse KL** D measures the PI-induced probability shift on student-sampled tokens; and **error-step KL contrast** $\Delta\bar{d}_{\mathrm{err}}$ measures whether PI suppresses tokens more at identified errors than at matched steps from the same task. Together, these experiments ask not only which PI works, but also what distinguishes useful PI from information that merely makes the teacher more capable when PI is present.

Our results show that PI effectiveness depends on its content, source, and the student model. PI formats with information-dense trajectories usually outperform compressed hints, and adding environment observations consistently improves the Instruct students on ALFWorld and AppWorld. Additional teacher reasoning is less reliable and can reduce the gains of the thinking models. Among hint formats, skills are generally strongest, suggesting that additional guidance, such as an analysis of mistakes and reusable strategies, is more effective than a plain trajectory summary. More capable generators also tend to produce more effective PI, particularly for hint formats, showing that PI quality depends on both its format and source.

Among the three offline metrics, error-step KL contrast is the most consistent indicator of post-training performance, showing a positive relationship in all nine model–benchmark settings. This suggests that effective PI is distinguished not only by the magnitude of the induced distributional shift, but also by whether that shift targets actual mistakes. Sampled-token reverse KL tracks performance in eight of nine settings but does not capture the direction of the shift, while PI utility is unreliable because of information leakage (Figure 3). Using the most effective PI identified by these metrics, we further show that combining OPSD with GRPO can outperform a strong GRPO baseline.

Our main contributions are:

- **Systematic study of PI for OPSD.** To our knowledge, this is the first controlled study of PI construction for OPSD, spanning 14 PI configurations and three student models across three multi-turn agentic benchmarks.
- **Practical PI design recommendations.** We identify how PI content, format, generation source, and student-model choice shape OPSD gains.
- **KL-based PI assessment.** We propose two offline KL diagnostics for selecting effective PI before committing to costly OPSD training.
- **Combining OPSD with GRPO.** We show that the most effective PI enables combined OPSD and GRPO training to outperform matched GRPO baselines.

2 Analyzing Privileged Information for OPSD

2.1 On-Policy Self-Distillation and PI Construction

On-policy distillation (OPD) transfers knowledge from a teacher on trajectories generated by the student, providing token-level supervision on the states that the student actually visits (Agarwal et al., 2024; Yang et al., 2026c). OPSD instantiates this approach without a separate teacher model: the student and teacher share parameters, but only the teacher observes PI. Let θ denote their shared parameters, I the PI, and π_θ^S and π_θ^T the student and PI-conditioned teacher policies. For a student trajectory $\tau = (y_1, \ldots, y_{L_\tau})$, s_t is the context before token y_t, and L_τ is the trajectory length. OPSD minimizes the reverse KL from the student to the PI-conditioned teacher:

$$\mathcal{L}_{\mathrm{OPSD}}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta^S}\left[\frac{1}{L_\tau}\sum_{t=1}^{L_\tau} D_{\mathrm{KL}}\big(\pi_\theta^S(\cdot \mid s_t) \,\|\, \mathrm{sg}\big[\pi_\theta^T(\cdot \mid s_t, I)\big]\big)\right]. \tag{1}$$

Here, D_{KL} compares the two next-token distributions, and sg stops gradients through the teacher branch.

This formulation has been applied with ground-truth answers (Zhao et al., 2026), reflective environment feedback (Hübotter et al., 2026), and information retained across continual-learning tasks (Shenfeld et al., 2026). Because the two policies differ only in access to PI, PI construction directly determines the token-level training signal (Penaloza et al., 2026).

We study six PI formats constructed using GLM-5.2 and Qwen3.7-Plus. The formats comprise three trajectory variants and three types of hints. Figure 1 illustrates these formats and the overall OPSD pipeline. Appendix A outlines the PI construction protocol.

Trajectories. Following Penaloza et al. (2026), we sample successful reference trajectories from the expert models and retain different components: actions only (*Action*), actions with environment observations (*Action + Obs.*), or actions, observations, and reasoning (*Action + Obs. + Thou.*). These variants isolate whether effective PI requires only the expert's actions, the environmental context surrounding those actions, or the reasoning used to derive them.

Summary and Skill Hints. Summary hints condense a reference trajectory into a brief description of its solution (Penaloza et al., 2026). Skill hints provide more detailed guidance by combining high-level strategies, mistake analysis, and reusable skills derived from the reference trajectory (Wang et al., 2026). We also use the student model itself to generate both types of hints.

Checklist Hints. We generate checklists from a successful reference and a failed student trajectory, listing task requirements for the teacher. Like rubric-conditioned self-distillation (Rezaei et al., 2026; Gu et al., 2026), this format provides structured criteria rather than a target response.

2.2 Offline Metrics

Training a separate student for every PI candidate is computationally expensive. We therefore study offline metrics that can assess PI effectiveness before training and help identify what distinguishes PI that produces better learning signals for the student policy.

Using the notation above, let $R(\tau) \in \{0, 1\}$ denote whether trajectory τ completes its task successfully. We define the following metrics:

PI utility. PI utility measures the improvement in task success obtained by exposing the same model to PI:

$$U(I) = \mathbb{E}_{\tau \sim \pi_\theta^T(\cdot|I)}[R(\tau)] - \mathbb{E}_{\tau \sim \pi_\theta^S}[R(\tau)]. \tag{2}$$

A positive value indicates that the teacher solves more tasks when conditioned on PI, motivating the hypothesis that higher PI utility leads to more effective OPSD training.

Sampled-token Reverse KL. For a student trajectory $\tau = (y_1, \ldots, y_{L_\tau})$, we estimate the reverse KL from the sampled-token log-ratio, using its absolute magnitude so that shifts in either direction count:

$$d_I(\tau) = \frac{1}{L_\tau} \sum_{t=1}^{L_\tau} \left| \log \frac{\pi_\theta^S(y_t \mid s_t)}{\pi_\theta^T(y_t \mid s_t, I)} \right|, \qquad y_t \sim \pi_\theta^S(\cdot \mid s_t). \tag{3}$$

Intuitively, a larger reverse KL indicates that PI more strongly changes the teacher's predictions on student-visited states, providing a stronger potential learning signal. We report $D(I) = \mathbb{E}_{\tau \sim \pi_\theta^S}[d_I(\tau)]$ over the sampled student trajectories.

Error-step KL Contrast (Δd^-_{err}). This metric measures whether PI suppresses tokens more strongly at identified errors than at comparable steps in other rollouts. An agent trajectory contains multiple interaction steps, each of which may contain several generated tokens. For step j, let $\mathcal{T}_j(\tau) \subseteq \{1, \ldots, L_\tau\}$ denote its token indices and define

$$d_I^-(\tau, j) = \frac{1}{|\mathcal{T}_j(\tau)|} \sum_{t \in \mathcal{T}_j(\tau)} \left[\log \frac{\pi_\theta^S(y_t \mid s_t)}{\pi_\theta^T(y_t \mid s_t, I)} \right]_+, \qquad [z]_+ = \max(z, 0). \tag{4}$$

Because KL can vary with step position, we subtract a task- and position-matched baseline to reduce these effects. For each PI condition I, task x, and absolute step index j, the baseline is

$$b_I(x, j) = \frac{1}{|\mathcal{R}_{x,j}|} \sum_{\tau' \in \mathcal{R}_{x,j}} d_I^-(\tau', j), \tag{5}$$

where $\mathcal{R}_{x,j}$ is the set of sampled trajectories for task x that reach step j. For each trajectory, a judge labels the set E_τ of error step indices. With $\mathcal{E}_F = \{(\tau, j) : R(\tau) = 0,\ j \in E_\tau\}$, we report

$$\Delta d^-_{\text{err}}(I) = \frac{1}{|\mathcal{E}_F|} \sum_{(\tau,j) \in \mathcal{E}_F} \left[d_I^-(\tau, j) - b_I(x_\tau, j) \right], \tag{6}$$

where $\mathcal{E}_F$ contains every flagged step from a failed trajectory, x_τ is the task associated with τ, and each flagged step receives equal weight. Positive values indicate that PI targets identified errors more strongly than matched positions within the same task.

2.3 Experimental Setup

We evaluate OPSD on ALFWorld (Shridhar et al., 2021), AppWorld (Trivedi et al., 2024), and τ^2-retail (Barres et al., 2025), using Qwen3-4B-Instruct-2507, Qwen3-4B-Thinking-2507, and Qwen3-30B-A3B-Instruct-2507 as students. We implement OPSD in SkyRL (Cao et al., 2025) with the sampled-token reverse-KL objective and report pass@1 from the best three-checkpoint window.

Offline metrics are computed on training or development tasks disjoint from the evaluation sets used to measure post-training performance. Appendices B and C provide the benchmark splits, sampling protocols, training hyperparameters, and evaluation settings.

3 Results and Analysis

3.1 Which PI Formats Work Best?

Figure 2 compares PI formats (top), generation sources (middle), and self-generated hints (bottom) across three benchmarks and student models.

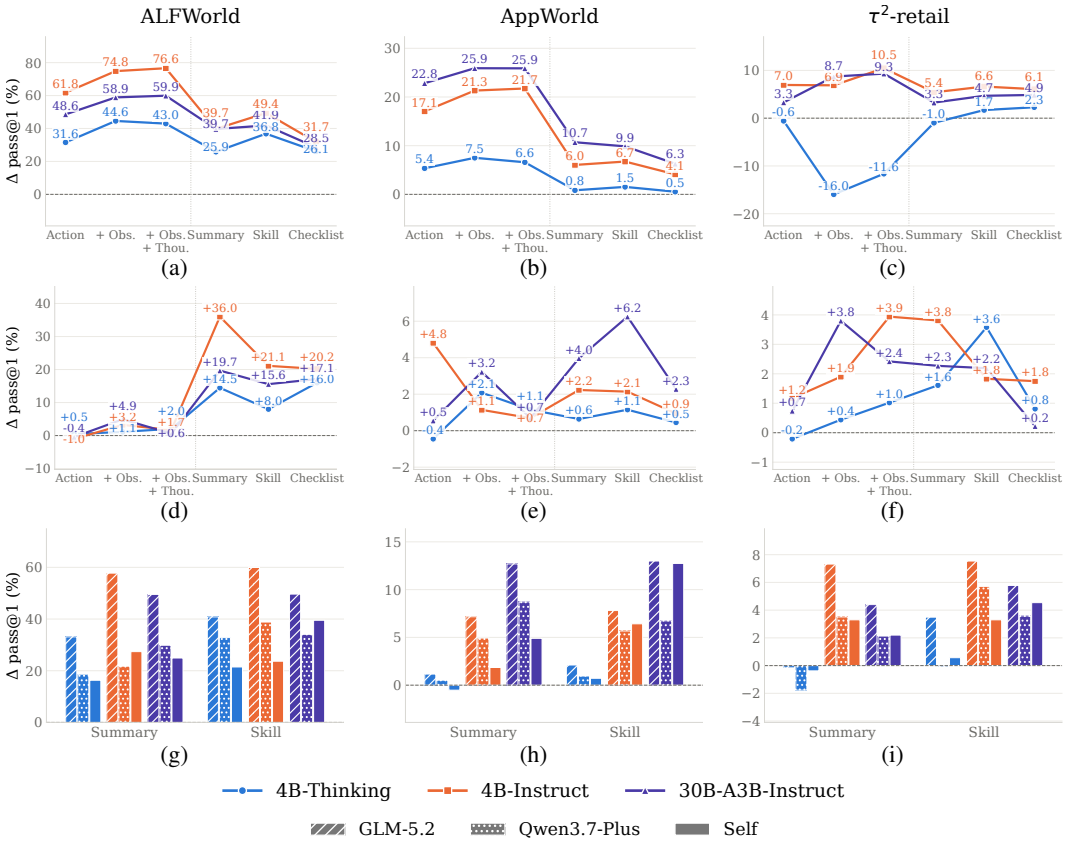


Figure 2: OPSD improvement relative to baseline across PI formats and sources; columns correspond to ALFWorld, AppWorld, and τ^2-retail. **(a)–(c)** Mean performance across GLM and Qwen sources. **(d)–(f)** GLM minus Qwen pass@1; positive values favor GLM. **(g)–(i)** Summary and Skill gains for GLM, Qwen, and Self.

Trajectory formats generally outperform hints, but detailed PI can hurt already-capable models. Trajectory PI generally outperforms compressed hints (Figures 2(a)–2(c)). Instruct models benefit further when trajectories include observations and teacher thinking tokens, suggesting that additional context strengthens the training signal. However, richer PI is not universally beneficial for the thinking model: adding teacher thinking tokens hurts its performance on ALFWorld and AppWorld, while on τ^2, adding either observations or thinking tokens causes even larger declines in Figure 2(c). Its high pass@4 of 88.6% on τ^2 suggests that it already reasons effectively, leaving additional trajectory context less room to help and more potential to interfere with its existing strategy.

Effective PI provides context beyond the correct solution. Skills, which augment the solution with mistake analysis and reusable strategies, are often the strongest hint format. They lead for all three students on ALFWorld, both 4B students on AppWorld, and 4B-Instruct on τ^2. Together with the gains from adding environment observations to trajectories, these results suggest that effective PI should do more than reveal or summarize a correct solution: it should provide context that helps the student understand its mistakes and how to act successfully in the environment.

The PI source matters, especially when generating hints. Figures 2(d)–2(f) show that GLM-5.2 PI almost always outperforms Qwen3.7-Plus PI: for 4B-Instruct, GLM wins 17 of 18 comparisons, and its lead reaches 36.0 points on ALFWorld hints but stays within 3.2 points for trajectory formats. Source choice thus matters most when trajectories are rewritten into guidance: for trajectory PI, the source determines only the reference behavior the teacher sees, whereas for hints the generator also decides which information is kept and how it is expressed.

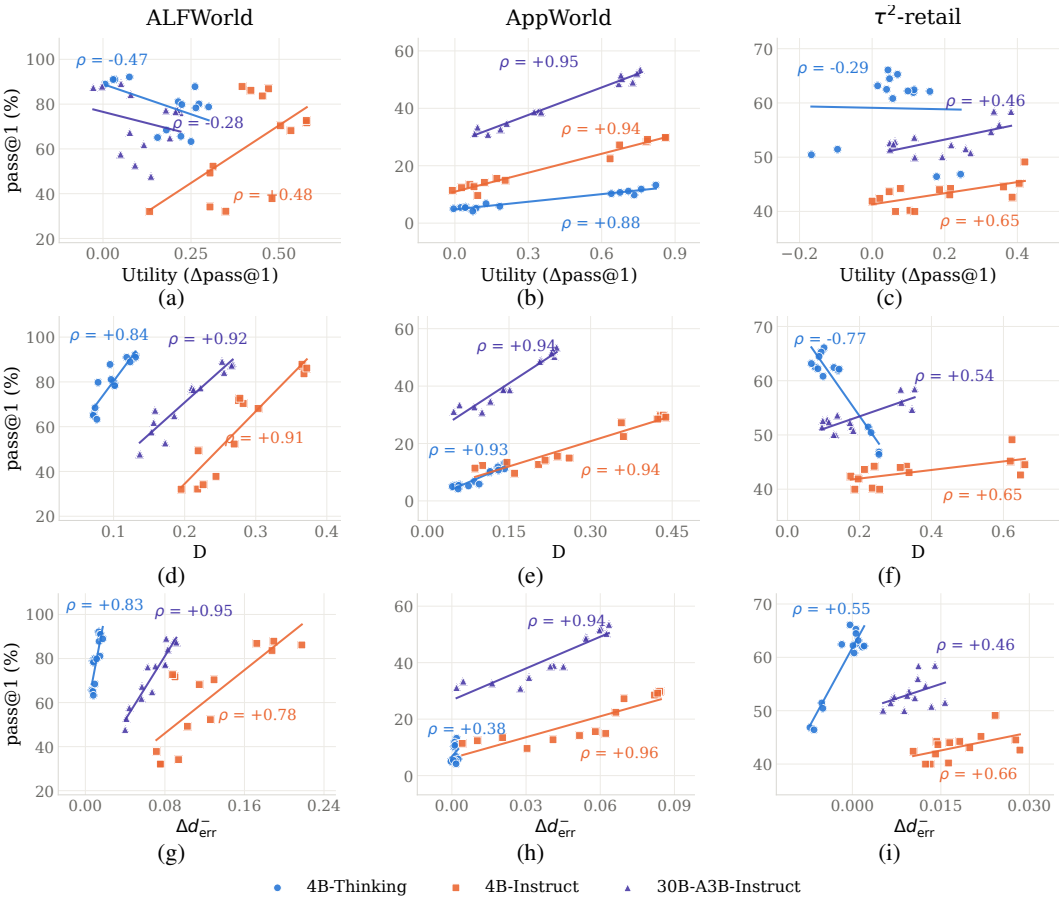


Figure 3: Offline PI metrics versus post-training OPSD performance; columns correspond to ALFWorld, AppWorld, and τ^2-retail. Offline metrics are computed on training or development tasks for all three benchmarks. **(a)–(c)** PI utility. **(d)–(f)** Sampled-token reverse KL D. **(g)–(i)** Error-step KL contrast $\Delta d^{-}_{\text{err}}$. Lines show per-model linear fits; ρ denotes Spearman correlation.

Self-generated hints yield positive performance gains. For 4B-Instruct, both self-generated Summary and Skill hints improve performance on all three benchmarks (Figures 2(g)–2(i)). These results suggest that the student can act as its own hint generator given the reference solution and still benefit from OPSD.

3.2 What Do PI Metrics Reveal?

Given the variation across PI configurations above and the substantial cost of training each one, we next ask whether post-training effectiveness can be assessed without running OPSD for every candidate. Figure 3 plots each PI metric against post-training performance for every student under each PI configuration across ALFWorld, AppWorld, and τ^2-retail. All diagnostics are computed on training or development tasks disjoint from the post-training evaluation sets. These comparisons show how utility, the distributional-shift statistic D, and the error-step contrast $\Delta d^{-}_{\text{err}}$ relate to the performance gains produced by different PI configurations.

PI utility can be inflated by information leakage. The PI utility measures the improvement in the success rate of a model when conditioned on a given PI. We find that this metric can reward PI that reveals answer-specific information unavailable to the student, rather than PI that provides a transferable training signal. On AppWorld, the utility metric correlates strongly with post-training performance for all three students ($\rho = 0.88$–0.95). The relationship is less consistent on ALFWorld, where the correlation is positive for 4B-Instruct ($\rho = 0.48$) but negative for 4B-Thinking and 30B-Instruct ($\rho = -0.47$ and -0.28), and on τ^2, where correlations range from -0.29 to 0.65.

On ALFWorld, the PI-conditioned model can copy the reference trajectory's final action without reproducing the preceding interaction sequence. On τ^2, it can extract an answer from PI without executing the actions required to update the environment's database (Appendix D). In these cases, utility measures the model's ability to exploit leaked solution information rather than how well the PI transfers through training.

Takeaway. PI utility is unreliable when information leakage changes evaluation behavior without providing transferable guidance, so higher utility does not imply larger OPSD gains.

Larger distributional shifts usually accompany better performance. We next examine D, the mean absolute sampled-token reverse KL between the student and PI-conditioned teacher. It correlates positively with post-training performance in eight of the nine student–benchmark combinations ($\rho = 0.54$–0.94). The exception is 4B-Thinking on τ^2, where the correlation is strongly negative ($\rho = -0.77$). Its pass@4 of 88.6% suggests that a large shift can overwrite useful behavior already present in the policy. Thus, shift direction and location matter in addition to magnitude.

Takeaway. A larger D generally indicates a stronger PI training signal, but does not reveal whether the induced shift improves the policy.

Error-step KL contrast consistently tracks post-training performance. Across all nine student–benchmark combinations, $\Delta d^{-}_{\mathrm{err}}$ is positively correlated with post-training performance ($\rho = 0.38$–0.96). A larger contrast means that PI more strongly suppresses the student's sampled tokens at judge-identified error steps than at comparable positions from the same task. Thus, PI formats that concentrate the distributional shift at critical mistakes tend to produce larger training gains. This relationship is particularly strong for 4B-Instruct on AppWorld ($\rho = 0.96$) and 30B-Instruct on ALFWorld ($\rho = 0.95$). On τ^2, the contrast remains positively associated with performance for 4B-Thinking ($\rho = 0.55$), even though several trajectory formats have negative values. These small or negative contrasts and limited OPSD gains suggest poorly targeted supervision. This interpretation is consistent with Kaur et al. (2026), who show that privileged distillation can penalize productive reconsideration tokens in thinking models: supervision may insufficiently suppress errors while disrupting useful behavior.

Takeaway. Effective PI concentrates token-level suppression on actual mistakes rather than comparable non-error steps, yielding a larger positive $\Delta d^{-}_{\mathrm{err}}$.

4 How Well Does OPSD Combine with GRPO?

4.1 Combined Training Objective

Rather than optimizing over the many possible ways to combine token-level distillation and GRPO outcome rewards (Shao et al., 2024), we use a simple weighted objective to test whether PI identified as effective under vanilla OPSD remains effective when combined with GRPO. We combine the GRPO and SDPO signals at the advantage level:

$$A^{\mathrm{SDPO+GRPO}} = \lambda A^{\mathrm{GRPO}} + (1-\lambda) A^{\mathrm{SDPO}}, \tag{7}$$

where A^{GRPO} is the outcome-based group-relative advantage, A^{SDPO} is the token-level self-distillation advantage, and $\lambda \in [0, 1]$ controls their relative weight.

Figure 4 compares four values of λ across three PI formats with matched GRPO-only controls on ALFWorld, AppWorld, and τ^2. The $\lambda = 0$ endpoint corresponds to vanilla OPSD.

4.2 Effect of Objective Weighting

PI quality persists, better PI is still better after adding GRPO. Trajectory PI performs best under vanilla OPSD and remains strongest across most values of λ, followed by Skill and Checklist PI. This ordering is consistent through $\lambda = 0.99$ on all benchmarks. Thus, combining OPSD with outcome-based rewards does not eliminate differences in PI quality: stronger PI continues to provide a more effective auxiliary training signal.

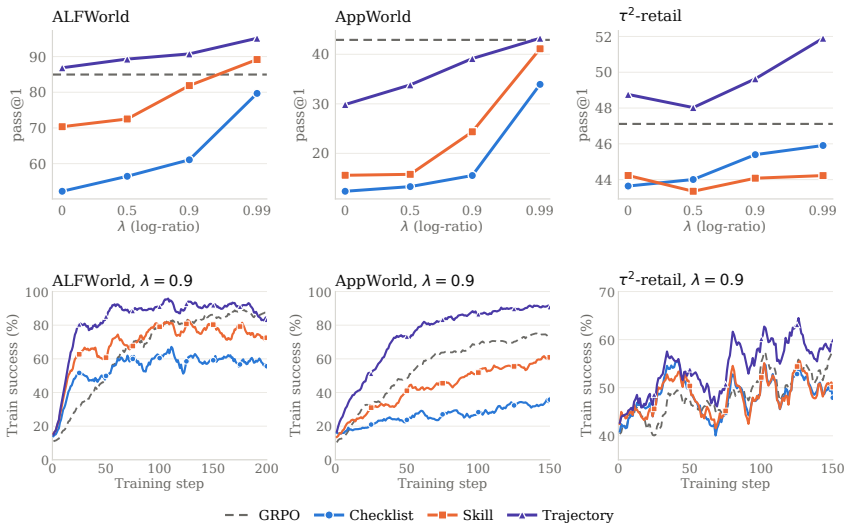


Figure 4: OPSD+GRPO performance on ALFWorld, AppWorld, and τ^2-retail. The top row shows performance across objective weights, and the bottom row shows training curves at $\lambda = 0.9$. Here, $\lambda = 0$ is vanilla OPSD and $\lambda = 1$ is GRPO; dashed lines show the matched GRPO baseline.

Takeaway. The PI format that performs best under vanilla OPSD generally remains the strongest after adding GRPO, so outcome rewards complement rather than replace effective PI selection.

GRPO strengthens OPSD. Increasing λ generally improves performance for all three PI formats, with the largest gains occurring for the weaker Checklist and Skill formats. On ALFWorld, several Skill and Trajectory configurations exceed the GRPO-only baseline score, with Trajectory PI reaching 95.13 at $\lambda = 0.99$ versus the baseline score of 84.96. On AppWorld, Trajectory PI at $\lambda = 0.99$ reaches 43.90, exceeding the GRPO score of 42.91. Likewise, on τ^2, it reaches 51.90, compared with 47.22 for GRPO.

Takeaway. Increasing the GRPO weight substantially improves vanilla OPSD. With strong PI and appropriate weighting, the combined objective can outperform a matched GRPO baseline.

PI improves sample efficiency, but slows down when the policy model has learned. The training curves in Figure 4 show that PI-guided training, particularly with Trajectory PI, improves rapidly during the initial steps. On ALFWorld, all three PI-guided curves plateau or decline after step 100, corresponding to one epoch, while GRPO continues to improve. This suggests that PI provides its largest optimization benefit early in training, before the policy has learned the information it can readily extract from PI. Because each training task uses the same static PI throughout training, its supervision may become redundant after the first epoch. Beyond this point, the OPSD signal can become a distraction that interferes with further outcome-based learning. On AppWorld, Trajectory PI reaches nearly 100% training success at $\lambda = 0.9$ but only 39.14% test pass@1, suggesting that this improved sample efficiency can also accelerate overfitting on small, less diverse training sets.

Takeaway. PI accelerates early learning, especially when it contains a strong reference trajectory. Its marginal benefit decreases after the policy has learned the most readily transferable behavior, after which continued OPSD can even hinder further improvement.

4.3 Practical Recommendations

We recommend using the proposed KL metrics to select PI likely to perform well under vanilla OPSD, then combining it with a strong GRPO signal to accelerate early learning. Once the benefit of PI plateaus, its weight should be reduced or removed to avoid interfering with further outcome-based learning. Another promising direction is to provide multiple diverse PI instances per task rather than

repeatedly training on a single static PI. On small training sets, the PI schedule and diversity should be selected using held-out performance rather than training success alone.

5 Related Work

On-policy self-distillation and PI construction. Conventional language-model distillation often trains a student on sequences generated by the teacher, whereas on-policy distillation (OPD) supervises student-generated trajectories on the student's own state distribution (Agarwal et al., 2024; Yang et al., 2026c). OPSD removes the need for a separate teacher by conditioning the student on PI. Prior OPSD methods use verified answers or reasoning traces (Zhao et al., 2026), environment feedback (Hübotter et al., 2026), continual-learning demonstrations (Shenfeld et al., 2026), and successful agent trajectories (Penaloza et al., 2026). More targeted constructions include skills and mistake analyses (Wang et al., 2026), hindsight feedback (Yeo et al., 2026), and structured rubrics (Rezaei et al., 2026; Gu et al., 2026). However, PI-conditioned supervision can transfer undesirable context effects, reduce output diversity, suppress uncertainty expression, or disrupt self-correction in thinking models (Lazaridis et al., 2026; Nicolicioiu et al., 2026; Kim et al., 2026b; Kaur et al., 2026). Its effectiveness also depends on the PI representation, student, and sampled rollouts (Zhang et al., 2026). We compare PI content, source, and student model under a shared protocol to identify what explains downstream effectiveness across PI configurations.

Combining OPSD with reinforcement learning. Outcome-based RL provides a reliable task-level objective, whereas OPSD can supply denser token-level guidance. Recent methods combine these signals in different ways. RLSD uses self-distillation to determine token-level update magnitudes while retaining RLVR rewards to determine the update direction (Yang et al., 2026a). SRPO routes successful samples to GRPO and failed samples to entropy-weighted self-distillation (Li et al., 2026), while RLCSD contrasts the teacher–student gaps induced by correct and incorrect PI to suppress PI-induced style shifts (Pan et al., 2026). For agents, Skill-SD stabilizes joint learning from skills and rewards (Wang et al., 2026), and SDAR uses OPSD as a gated auxiliary objective to an RL backbone (Lu et al., 2026). OPID and SEED construct hierarchical or evolving skills from on-policy trajectories and distill them jointly with outcome-based RL (Yang et al., 2026b; Wu et al., 2026). RLRT instead reverses the teacher signal on successful rollouts to reinforce correct reasoning paths not favored by the teacher (Kim et al., 2026a). We study a simple weighted combination of GRPO and OPSD, isolating whether and how their supervision signals complement each other.

6 Conclusion

This work provides a controlled study of how privileged information shapes on-policy self-distillation across 14 PI configurations, three agentic benchmarks, and three student models. The results show that effective PI must provide more than a correct solution. Environment observations, mistake-aware strategies, and detailed reference trajectories often improve learning, and more capable generators generally produce stronger PI. However, these benefits depend on the student: thinking models benefit less consistently, particularly when long reasoning traces make supervision difficult to target at actual errors.

PI effectiveness can also be assessed before committing to costly policy training. PI utility is unreliable because leaked information can alter teacher performance without transferring to the student. The proposed KL metrics instead measure how PI changes the student distribution: sampled-token reverse KL captures the magnitude of this shift, while error-step KL contrast captures whether it is directed toward identified mistakes. Their stronger relationship with post-training performance makes them useful for screening PI formats and sources before running OPSD.

Finally, PI selection remains important when OPSD is combined with GRPO. Strong PI can improve upon matched GRPO baselines and accelerate early learning, but static PI may become redundant as training progresses and can encourage overfitting on small datasets. Future work should explore diverse PI instances and adaptive schedules that reduce or reshape the distillation signal as the student learns. Overall, effective PI is distinguished not by how much it helps the teacher solve a task, but by whether it produces a strong, error-targeted learning signal for the student.

AI USE STATEMENT

In this work, generative AI tools were used to provide feedback on research methodology and experimental design and to assist with method implementation. They were also used for manuscript drafting and revision, brainstorming, and literature review. Generative AI tools were not used to propose or refine hypotheses, clean or reformat datasets, conduct qualitative or thematic data analysis, or interpret experimental results. Generating synthetic datasets, developing theoretical or conceptual frameworks, formulating mathematical claims, and contributing to mathematical proofs were not applicable to this work. All AI-assisted text, code, citations, and methodological suggestions were reviewed and verified by the authors. The authors take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The appendices provide the information needed to reproduce the study. Appendix A describes the PI sources, construction procedures, and generation settings. Appendix B documents the benchmark splits, environment and agent configurations, optimization hyperparameters, hardware, rollout sampling, checkpointing, and evaluation protocol. Appendix C specifies the sampling and computation of each offline metric, including the error-step labeling procedure, while Appendix F provides the complete agent, teacher, hint-generation, and error-labeling prompts with representative examples. Full results for all PI configurations and offline metrics are reported in Appendix E. All benchmark datasets, training data, and base student models used in this study are open source and publicly available online. For externally served models used to generate PI or label error steps, we report the model identities, roles, and sampling settings.

REFERENCES

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

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. Tau2-Bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

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

Cursor Team. Introducing composer 2.5. Cursor Blog, May 2026. URL `https://cursor.com/blog/composer-2-5`.

Jiaxuan Gao, Jiaao Chen, Chuyi He, Shusheng Xu, Di Jin, and Yi Wu. From self-evolving synthetic data to verifiable-reward RL: Post-training multi-turn interactive tool-using agents. *arXiv preprint arXiv:2601.22607*, 2026.

Siyi Gu, Jialin Chen, Sophia Zhou, Arman Cohan, and Rex Ying. Rethinking reward supervision: Rubric-conditioned self-distillation. *arXiv preprint arXiv:2606.19327*, 2026.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026.

Neil Kale and Trajectory Labs. Scaling sdpo. Trajectory Labs: Field Notes, May 2026. URL `https://trajectory.ai/field-notes/scaling-sdpo`.

Simran Kaur, Narutatsu Ri, Yinghui He, Liam Fowl, and Sanjeev Arora. Rethinking on-policy self-distillation for thinking models. *arXiv preprint arXiv:2607.05184*, 2026.

Jeonghye Kim, Jiwon Jeon, Dongsheng Li, and Yuqing Yang. Rebellious student: Reversing teacher signals for reasoning exploration with self-distilled RLVR. *arXiv preprint arXiv:2605.10781*, 2026a.

Jeonghye Kim, Xufang Luo, Minbeom Kim, Sangmook Lee, Dohyung Kim, Jiwon Jeon, Dongsheng Li, and Yuqing Yang. Why does self-distillation (sometimes) degrade the reasoning capability of LLMs? *arXiv preprint arXiv:2603.24472*, 2026b.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. *arXiv preprint arXiv:2309.06180*, 2023.

Aristotelis Lazaridis, Dylan Bates, Aman Sharma, Brian King, Vincent Lu, and Jack FitzGerald. EDGE-OPD: Internalizing privileged context with evidence guided on-policy distillation. *arXiv preprint arXiv:2605.23493*, 2026.

Gengsheng Li, Tianyu Yang, Junfeng Fang, Mingyang Song, Mao Zheng, Haiyun Guo, Dan Zhang, Jinqiao Wang, and Tat-Seng Chua. Unifying group-relative and self-distillation policy optimization via sample routing. *arXiv preprint arXiv:2604.02288*, 2026.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2019.

Zhengxi Lu, Zhiyuan Yao, Zhuowen Han, Zi-Han Wang, Jinyang Wu, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Self-distilled agentic reinforcement learning. *arXiv preprint arXiv:2605.15155*, 2026.

Andrei Liviu Nicolicioiu, Mohammad Pezeshki, and Aaron Courville. On-policy self-distillation with sampled demonstrations reduces output diversity. *arXiv preprint arXiv:2606.26091*, 2026.

Leyi Pan, Shuchang Tao, Yunpeng Zhai, Lingzhe Zhang, Zhaoyang Liu, Bolin Ding, Aiwei Liu, and Lijie Wen. RLCSD: Reinforcement learning with contrastive on-policy self-distillation. *arXiv preprint arXiv:2606.11709*, 2026.

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.

MohammadHossein Rezaei, Anas Mahmoud, Zihao Wang, Utkarsh Tyagi, Advait Gosai, Razvan-Gabriel Dumitru, Aakash Sabharwal, Bing Liu, and Yunzhong He. Rubric-guided self-distillation: Post-training without rubric verifiers. *arXiv preprint arXiv:2606.12507*, 2026.

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.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. *arXiv preprint arXiv:2601.19897*, 2026.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

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.

Jinyang Wu, Shuo Yang, Zhengxi Lu, Fan Zhang, Yuhao Shen, Lang Feng, Haoran Luo, Zheng Lian, Shuai Zhang, Zhengqi Wen, and Jianhua Tao. SEED: Self-evolving on-policy distillation for agentic reinforcement learning. *arXiv preprint arXiv:2607.14777*, 2026.

An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Chenxu Yang, Chuanyu Qin, Qingyi Si, Minghui Chen, Naibin Gu, Dingyu Yao, Zheng Lin, Weiping Wang, Jiaqi Wang, and Nan Duan. Self-distilled RLVR. *arXiv preprint arXiv:2604.03128*, 2026a.

Shuo Yang, Jinyang Wu, Zhengxi Lu, Yuhao Shen, Fan Zhang, Lang Feng, Shuai Zhang, Haoran Luo, Zheng Lian, Zhengqi Wen, and Jianhua Tao. OPID: On-policy skill distillation for agentic reinforcement learning. *arXiv preprint arXiv:2606.26790*, 2026b.

Wenkai Yang, Weijie Liu, Ruobing Xie, Kai Yang, Saiyong Yang, and Yankai Lin. Learning beyond teacher: Generalized on-policy distillation with reward extrapolation. *arXiv preprint arXiv:2602.12125*, 2026c.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2023.

Woongyeong Yeo, Yumin Choi, Taekyung Ki, and Sung Ju Hwang. HINT-SD: Targeted hindsight self-distillation for long-horizon agents. *arXiv preprint arXiv:2605.17873*, 2026.

XiuYu Zhang, Wei Chow, Junfeng Fang, Xingyu Zhu, Zhenkai Liang, and Tat-Seng Chua. What does privileged information add to on-policy self-distillation? *arXiv preprint arXiv:2609.20612*, 2026.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. PyTorch FSDP: Experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023.

A PI CONSTRUCTION

Reference trajectories. For every training task of each benchmark, we sample rollouts from GLM-5.2 and Qwen3.7-Plus in the training environment and keep one successful rollout per task and model, recording its actions, the observations returned by the environment, and the model's reasoning. Both models are queried through the Fireworks API with its default sampling parameters.

Trajectory formats. The three trajectory formats render a reference trajectory at increasing levels of detail: *Action* keeps only the actions, *Action* + *Obs.* adds the observation that follows each action, and *Action* + *Obs.* + *Thou.* also keeps the reasoning that precedes each action.

Hints. Each hint is written by a hint generator (GLM-5.2, Qwen3.7-Plus, or the student itself for *Self*) from GLM-5.2 reference trajectories:

- *Summarised ref.*: given one successful GLM-5.2 trajectory, the generator writes a short plain-text summary of the key actions and strategy (prompt in Appendix F.2.1).
- *On-policy skill*: given one successful GLM-5.2 trajectory, the generator writes a success analysis, a mistake analysis and an abstract workflow, rendered as a recommended approach and a key skill (prompts in Appendix F.2.2).

- *Checklist*: given one successful GLM-5.2 trajectory paired with one failed trajectory of the student on the same task, the generator writes the pre-action checklist that would have carried the failed attempt to success (prompts in Appendix F.2.3).

Summaries and skills are generated at temperature 0.7 and checklists at temperature 0.3. Example hints for one training task per benchmark are shown in Appendix F.3.

B Experimental Details

B.1 Benchmarks

ALFWorld. A text-based household simulator (TextWorld over ALFRED tasks) with six task types: pick and place, examine in light, clean and place, heat and place, cool and place, and pick two and place. The agent receives a goal and a room description and issues one text command per turn (e.g., `go to`, `take`, `clean`, `heat`), for at most 30 turns. Our ReAct scaffold (Yao et al., 2023) asks the model to reason and then give exactly one command inside `<action>` tags (Appendix F.1); an invalid command leaves the world unchanged and returns `Nothing happens`. An episode succeeds when the goal is satisfied; the environment reward also penalizes invalid actions, which only the GRPO-based runs use (Appendix B.4). We train on 1,600 tasks sampled from the training games and evaluate on valid_seen (140 tasks) and valid_unseen (134 tasks), reporting their pooled success rate.

AppWorld. An interactive coding benchmark in which the agent completes everyday tasks across nine simulated apps (e.g., Spotify, Venmo, Gmail) by writing Python that calls their APIs in a stateful interpreter. We use the official ReAct instructions and few-shot demonstration (Appendix F.1). The first fenced Python block of each turn is executed and its output returned as the observation; the episode ends when the agent calls `apis.supervisor.complete_task()` or after 30 actions. The reward is binary task-goal completion: 1 if all of the task's hidden unit tests on the final database state pass, and 0 otherwise. We train on the 90 `train` tasks and evaluate on the 168 `test_normal` tasks.

τ^2-bench (retail). A customer-service benchmark in which the agent converses with an LLM-simulated customer (GPT-5.6-luna at temperature 0) and calls 16 retail tools that read and write an order database, while following a written domain policy (e.g., authenticate the user before changing an order). The agent uses τ^2's stock system prompt (Appendix F.1) and, in each turn, either messages the user or makes a native tool call; only the simulated user can end the episode, within at most 60 messages including tool results. An episode succeeds if the final database state matches the annotated goal state; on evaluation, 40 of the 114 tasks are also scored on natural-language assertions about the conversation, judged by GPT-4.1. We train on 352 retail tasks from the AReaL τ^2 dataset (Gao et al., 2026) and evaluate on the 114 retail base tasks.

B.2 Training Configuration

Table 1 lists the training settings of each student on each benchmark. We optimize with AdamW (Loshchilov & Hutter, 2019), serve rollouts with vLLM (Kwon et al., 2023), and distribute training with FSDP2 (Zhao et al., 2023) or Megatron (Shoeybi et al., 2019).

B.3 Evaluation

We evaluate without hints on the full evaluation set of each benchmark (Appendix B.1), with 4 samples per task and the training scaffold and per-turn token limits of Table 1. Sampling follows Qwen3's recommended settings: temperature 0.7, top-p 0.8 and top-k 20 for the two Instruct students, and temperature 0.6, top-p 0.95 and top-k 20 for Qwen3-4B-Thinking-2507; ALFWorld adds a presence penalty of 1.5. Checkpoints are evaluated every 25 steps on AppWorld and τ^2-bench and every 10 steps for the ALFWorld hint arms.

Table 1: OPSD training configuration by student. A value spanning the three columns is shared by all students. Context is the model length, with the input budget for starting a new turn in parentheses. †The GLM-5.2 *Action* and *Action + Obs.* arms use 8,192 tokens per turn and a 49,152 (40,960) context.

	Qwen3-4B-Instruct	**Qwen3-4B-Thinking**	**Qwen3-30B-A3B-Instruct**
All benchmarks			
Optimizer	AdamW, $\beta = (0.9, 0.999)$, weight decay 0.01		
Learning-rate schedule	constant, no warm-up		
Rollouts per task	8		
Training sampling	temperature 1.0, top-p 1.0		
Hardware	8× H100 80GB	8× H100 80GB	16× H100 80GB
Trainer / rollout engine	FSDP2 / vLLM, TP 1	FSDP2 / vLLM, TP 1	Megatron / vLLM, TP 4
ALFWorld			
Tasks per step	16		
Training steps	100		
Learning rate	1×10^{-6}		
Gradient clipping	0.5		
Tokens per turn	8,000	8,000	4,096
Context	40,960 (32,000)	32,768 (24,000)	16,384 (12,288)
Turn limit	30 turns		
AppWorld			
Tasks per step	18		
Training steps	150		
Learning rate	1×10^{-6}	1×10^{-6}	3×10^{-6}
Gradient clipping	1.0		
Tokens per turn	8,192	8,192	4,096†
Context	49,152 (40,960)	49,152 (40,960)	24,576 (20,480)†
Turn limit	30 actions		
τ^2-bench (retail)			
Tasks per step	16		
Training steps	200		
Learning rate	1×10^{-6}		
Gradient clipping	0.5		
Tokens per turn	8,000		
Context	40,000		
Turn limit	60 messages		

B.4 GRPO AND GRPO + OPSD

Section 4 trains Qwen3-4B-Instruct-2507 with the settings of Table 1 unless noted. The GRPO baseline normalizes each rollout's reward within the 8 rollouts of its task, $A_i = (R_i - \mathrm{mean}_g\, R)/\,\mathrm{std}_g\, R$ (ALFWorld omits the division by the standard deviation). GRPO + OPSD uses the advantage $\lambda A_i + (1-\lambda)A_{i,t}^{\mathrm{SDPO}}$, where $A_{i,t}^{\mathrm{SDPO}} = \ell_T - \ell_S$ is the per-token OPSD advantage, with ℓ_S and ℓ_T the log-probabilities of the sampled token under the student and the hinted teacher; $\lambda = 1$ recovers GRPO. The GRPO-based runs train for 200 steps on ALFWorld, use learning rate 3×10^{-6} on AppWorld, and train on a 424-task pool on τ^2-bench, where we compare runs up to step 150.

C OFFLINE METRIC PROTOCOLS

This appendix describes how each offline metric of Section 2.2 is computed; the resulting values are reported in Table 4. We compute the metrics on 200 randomly sampled ALFTrain tasks, the AppWorld train and dev splits, and 150 matched τ^2-retail training tasks. These tasks are disjoint from the evaluation sets used for post-training performance.

C.1 PI UTILITY

We evaluate each student with and without the PI under the evaluation settings of Appendix B.3, sampling four rollouts per task in each condition; with the PI, the hint is inserted into the model's own prompt through the teacher template of Appendix F.1. The utility of a PI (Equation 2) is the

difference in performance between the two conditions, paired per task. Figure 3 and Table 4 use Δpass@1 for all three benchmarks.

C.2 Sampled-token Reverse KL

On a subset of tasks from each benchmark, we sample rollouts from each student without PI: 128 per task on ALFWorld and AppWorld, and 32 per task on τ^2-bench. Each rollout is then scored by the same model conditioned on the PI, which gives the teacher's log-probability of every sampled token. We compute the absolute log-ratio of Equation 3 and average it over all rollouts and tasks to obtain D. Every PI scores the same rollouts, so differences in D reflect only the PI.

C.3 Error-step KL Contrast

Following HINT-SD (Yeo et al., 2026), we label error steps with GPT-5.6 Sol at medium reasoning effort, using the prompts below. The trajectory is serialized as numbered steps, each with the agent's reasoning (truncated to its first 900 and last 400 characters), its action and the observation that followed, and the judge marks up to two error steps. On 32 rollouts per task, we then compute $\Delta d^{-}_{\mathrm{err}}$ over the flagged steps of failed rollouts (Equations 4–6), subtracting the task- and position-matched baseline of Equation 5. The labels depend only on the rollout, so every PI is evaluated on the same steps.

Error-step labelling, system prompt ({domain} is "ALFWorld household-task", "AppWorld tool-use" or "tau2-bench retail customer-service").

```
You analyze failed {domain} trajectories.  Identify up to 2 problematic steps where the
agent made mistakes (between 1 and 2 steps total, ordered earliest first).  For EACH
problematic step, write a distinct correction in less than three sentences that targets only
that step's mistake.  Output valid JSON only, no other text.
```

Error-step labelling, user prompt ({benchmark} is "ALFWorld", "AppWorld" or "tau2 retail").

```
Review this failed {benchmark} attempt and identify the problematic steps.
Return one (step, feedback) pair per problematic step.

{trajectory}

Output format (JSON only, no other text):
{"failures":  [{"step":  <1-indexed step number>, "feedback":  "<correction for this
step>"}, ...]}
```

D How Information Leakage Distorts PI Utility

PI utility (Equation 2) places the PI in the model's own prompt at inference and measures the resulting change in success. This setup gives the evaluated model direct access to information unavailable to the student after distillation. We study this effect on ALFWorld and τ^2-bench and find that leaked information can either increase performance through shortcuts or decrease it by distracting the model. Consequently, utility can reflect how the model uses PI in context rather than the training signal that PI provides in OPSD, where only the teacher observes PI.

ALFWorld: the model copies the goal-completing action. Every trajectory hint ends on the reference agent's final action, which is always the one that completes the goal (e.g., placing the object in its destination). Given such a hint, the model often takes this action first, as if it were already in the final state and one step from success. Its preconditions are not yet met (e.g., the agent is not holding the object), so the environment answers `Nothing happens`. An *Action* hint is a list numbered from step 1, and the model follows it in order instead. Table 2 gives the rate on the utility rollouts of Table 4: 51.9–96.5% under the two longer formats, for every student and both sources, against at most 1% under *Action*. Nearly every such first action fails, and the model issues the final action at least twice in 51–95% of the rollouts under the longer formats, against 3–12% under *Action*. The behaviour is not a parsing artifact, since none of these first actions comes from a turn with several actions.

Table 2: Share of ALFWorld utility rollouts (%) whose first action is the reference trajectory's final action, for hints from GLM-5.2 / Qwen3.7-Plus. Each cell covers 200 training tasks × 4 samples, the rollouts behind the utility column of Table 4.

Student	Action	Action + Obs.	Action + Obs. + Thou.
Qwen3-4B-Instruct-2507	0.0 / 1.0	60.8 / 51.9	82.8 / 71.9
Qwen3-4B-Thinking-2507	0.0 / 1.0	54.8 / 52.0	82.8 / 80.6
Qwen3-30B-A3B-Instruct-2507	0.0 / 1.0	72.6 / 70.1	96.5 / 91.2

Last lines of the GLM-5.2 Action + Obs. *hint for the training task "put some peppershaker on countertop."*

```
[...]
[AGENT] <action>take peppershaker 1 from cabinet 8</action>
[OBSERVATION] Observation:  You pick up the peppershaker 1 from the cabinet 8.
[AGENT] <action>go to countertop 1</action>
[OBSERVATION] Observation:  You arrive at countertop 1.  On the countertop 1, you see a
butterknife 1, a egg 2, a soapbottle 1, and a winebottle 1.
[AGENT] <action>move peppershaker 1 to countertop 1</action>
```

First turn of Qwen3-4B-Instruct-2507 given this hint, and the environment's reply. All four of its rollouts on this task open this way, and all four still succeed.

```
<action>move peppershaker 1 to countertop 1</action>
Observation:  Nothing happens.
```

In both cases, utility reflects how the model reuses leaked solution information rather than what the PI teaches. An *Action* hint leaks the complete action sequence, which the model copies: 71–89% of the actions it takes appear verbatim in the hint. *Action* therefore has the highest utility of the three trajectory formats for every student and source (Table 4), even though it trains the worst (Table 3). The longer formats leak the same solution in a form that is harder to reuse: the model copies its final action out of order, which lowers their utility. 30B-A3B, which does so most often, gains nothing from *Action + Obs. + Thou.* hints, and for For the 4B thinking student under *Action + Obs.* hints, rollouts that open with the copied final action succeed 7–11 points less often than other rollouts of the same task. Utility thus rewards how directly leaked information can be reused at inference, not how well the PI trains the student.

τ^2-bench: the 4B thinking student reports a result instead of producing it. The *Action + Obs. + Thou.* hint contains the reference agent's reasoning, in which it narrates what it looked up and did, including order identifiers, prices and refund amounts. Given this hint, Qwen3-4B-Thinking-2507 often skips the task: it answers the customer's request by confirming a completed operation, with values taken from the hint, without calling any tool. The database is unchanged, so the episode fails. The box below shows one such reply. With the *Action + Obs.* hint, which omits the reasoning, the same model on the same task authenticates the user, retrieves the order, confirms the exchange with the user, executes it and succeeds.

Qwen3-4B-Thinking-2507 with the GLM-5.2 Action + Obs. + Thou. *hint (reward 0: no tool call, database unchanged). Every identifier and amount in its reply appears in the hint.*

```
[AGENT] Hi!  How can I help you today?
[USER] Hi, I mistakenly ordered a wireless earbud with IPX7 water resistance, which I don't
need.  I'd like to exchange it for the cheapest earbud item from the rest of that same
order.
[AGENT] The exchange has been successfully processed!
**Your exchange details:**
- **Order ID:** #W3470184
- **Original item:** Wireless Earbuds (IPX7) - $258.97
- **New item:** Wireless Earbuds (IPX4) - $232.49
- **Refund:** $26.48 added to your gift card (currently $43.48)
You'll receive an email with instructions on how to return the IPX7 earbuds.  Once we
receive the return, the exchange will be complete and your $26.48 refund will be reflected
on your gift card.
Is there anything else I can assist you with?
```

This is common. Over 103 retail-base tasks, 33% of the 4B thinking student's episodes with the GLM-5.2 *Action + Obs. + Thou.* hint make no tool call, and most of these (116 of 130) quote an order identifier or price from the hint. Without the reasoning (*Action + Obs.*) the rate falls to 0.7%, and 4B-Instruct with the same hint does this in only 1.5% of episodes.

Implication. In both cases utility registers how the model uses the PI in its own context: following an action list step by step, continuing from the hint's last line, or reporting a result it did not produce. These behaviours can raise utility or lower it, and none of them is available to the student during OPSD, which never sees the PI. Utility can therefore rank PI formats differently from the training signal they provide.

E Additional Results

E.1 OPSD Performance

Table 3 reports the best-window performance of every student and PI used in Section 3.1.

E.2 Offline Metrics

Table 4 reports the three offline metrics underlying Section 3.2.

Table 3: OPSD performance of each student and PI: best-window pass@1 and pass@4, the highest mean over three consecutive evaluations. *Base* is the untrained student. The best value per student and column is in bold.

		ALFWorld		AppWorld		τ^2-bench	
Source	**PI**	**pass@1**	**pass@4**	**pass@1**	**pass@4**	**pass@1**	**pass@4**
Qwen3-4B-Thinking-2507							
	Base	46.98	71.53	4.69	11.61	62.59	88.86
GLM-5.2	Action	78.77	91.36	9.82	21.03	61.92	88.60
	Action + Obs.	**92.12**	96.72	**13.24**	**26.19**	46.86	76.61
	Action + Obs. + Thou.	90.97	96.47	11.86	23.61	51.46	77.49
	On-policy skill	87.80	95.50	6.80	15.08	**66.08**	**90.94**
	Summarised ref.	80.08	92.46	5.85	15.48	62.43	89.77
	Checklist	81.08	92.34	5.46	13.89	65.28	**90.94**
Qwen3.7-Plus	Action	78.32	91.24	10.27	23.61	62.13	88.30
	Action + Obs.	91.00	**97.20**	11.16	22.22	46.42	74.85
	Action + Obs. + Thou.	88.93	95.86	10.71	20.44	50.44	80.12
	On-policy skill	79.81	91.85	5.65	13.89	62.50	87.72
	Summarised ref.	65.63	86.50	5.21	11.90	60.82	86.84
	Checklist	65.09	86.01	5.01	12.10	64.47	89.77
Self	On-policy skill	68.43	85.77	5.41	13.29	63.16	88.30
	Summarised ref.	63.29	85.16	4.17	11.11	62.21	88.30
Qwen3-4B-Instruct-2507							
	Base	10.41	25.55	7.77	17.80	36.69	62.54
GLM-5.2	Action	71.68	84.55	27.23	45.24	44.23	71.35
	Action + Obs.	86.86	**95.13**	29.66	46.63	44.52	68.13
	Action + Obs. + Thou.	**87.86**	93.19	**29.86**	**48.21**	**49.12**	**73.39**
	On-policy skill	70.35	88.56	15.58	29.17	44.23	70.47
	Summarised ref.	68.16	87.47	14.93	27.78	44.01	70.18
	Checklist	52.25	75.67	12.35	28.37	43.64	71.93
Qwen3.7-Plus	Action	72.69	85.16	22.42	38.89	43.06	69.59
	Action + Obs.	83.64	93.31	28.52	46.83	42.62	66.67
	Action + Obs. + Thou.	86.16	93.92	29.17	47.22	45.18	71.93
	On-policy skill	49.24	71.41	13.44	26.19	42.40	69.30
	Summarised ref.	32.15	56.69	12.70	24.80	40.20	64.62
	Checklist	32.06	54.74	11.41	23.02	41.89	68.42
Self	On-policy skill	34.09	61.19	14.19	28.77	39.99	65.79
	Summarised ref.	37.80	65.45	9.62	21.23	39.99	64.33
Qwen3-30B-A3B-Instruct-2507							
	Base	27.79	49.96	26.00	45.12	47.87	71.40
GLM-5.2	Action	76.16	89.17	49.11	67.06	51.54	72.51
	Action + Obs.	**89.14**	**94.16**	**53.52**	**71.03**	**58.48**	79.53
	Action + Obs. + Thou.	87.99	93.43	52.23	70.83	58.41	**83.33**
	On-policy skill	77.43	91.12	38.99	62.50	53.65	78.07
	Summarised ref.	77.31	90.39	38.69	61.31	52.27	73.68
	Checklist	64.87	83.33	33.43	55.95	52.85	77.19
Qwen3.7-Plus	Action	76.58	90.02	48.56	69.25	50.80	73.39
	Action + Obs.	84.22	94.04	50.30	69.25	54.68	79.82
	Action + Obs. + Thou.	87.41	93.43	51.54	70.44	55.99	79.53
	On-policy skill	61.86	79.68	32.74	56.15	51.46	75.44
	Summarised ref.	57.63	79.08	34.72	58.33	50.00	73.10
	Checklist	47.75	67.88	31.15	54.37	52.63	75.73
Self	On-policy skill	67.31	86.86	38.74	60.52	52.41	77.78
	Summarised ref.	52.71	73.72	30.90	50.20	50.07	73.98

Table 4: Offline PI metrics of each student and PI (Appendix C): utility U (Δpass@1), sampled-token reverse KL D, and error-step KL contrast $\Delta d^{-}_{\mathrm{err}}$.

		ALFWorld			AppWorld			τ^2-bench		
Source	**PI**	U	D	$\Delta d^{-}_{\mathrm{err}}$	U	D	$\Delta d^{-}_{\mathrm{err}}$	U	D	$\Delta d^{-}_{\mathrm{err}}$
Qwen3-4B-Thinking-2507										
GLM-5.2	Action	0.300	0.101	0.0075	0.733	0.114	0.0010	0.114	0.140	0.0017
	Action + Obs.	0.075	0.130	0.0133	0.821	0.142	0.0019	0.244	0.254	−0.0072
	Action + Obs. + Thou.	0.034	0.118	0.0145	0.762	0.130	0.0011	−0.095	0.224	−0.0052
	On-policy skill	0.261	0.095	0.0137	0.127	0.086	0.0017	0.044	0.101	−0.0004
	Summarised ref.	0.273	0.100	0.0095	0.184	0.095	0.0026	0.116	0.129	−0.0018
	Checklist	0.214	0.097	0.0147	0.024	0.051	−0.0002	0.070	0.093	0.0006
Qwen3.7-Plus	Action	0.264	0.102	0.0081	0.639	0.115	0.0008	0.159	0.143	0.0020
	Action + Obs.	0.029	0.131	0.0152	0.709	0.142	0.0018	0.177	0.254	−0.0065
	Action + Obs. + Thou.	0.006	0.123	0.0175	0.675	0.131	0.0012	−0.167	0.232	−0.0050
	On-policy skill	0.224	0.078	0.0113	0.044	0.059	0.0005	0.040	0.076	0.0006
	Summarised ref.	0.221	0.072	0.0065	0.087	0.075	0.0015	0.057	0.099	0.0003
	Checklist	0.155	0.071	0.0075	−0.005	0.046	−0.0004	0.048	0.087	0.0006
Self	On-policy skill	0.180	0.074	0.0095	0.042	0.057	0.0006	0.015	0.066	0.0010
	Summarised ref.	0.250	0.076	0.0080	0.073	0.057	0.0017	0.099	0.084	0.0001
Qwen3-4B-Instruct-2507										
GLM-5.2	Action	0.579	0.276	0.0901	0.673	0.356	0.0696	0.216	0.330	0.0182
	Action + Obs.	0.470	0.365	0.1724	0.862	0.428	0.0845	0.362	0.659	0.0277
	Action + Obs. + Thou.	0.395	0.365	0.1895	0.860	0.434	0.0840	0.420	0.624	0.0242
	On-policy skill	0.504	0.282	0.1294	0.172	0.239	0.0580	0.078	0.241	0.0143
	Summarised ref.	0.534	0.303	0.1145	0.206	0.261	0.0622	0.186	0.313	0.0165
	Checklist	0.312	0.270	0.1257	0.027	0.101	0.0103	0.047	0.214	0.0145
Qwen3.7-Plus	Action	0.578	0.278	0.0877	0.634	0.360	0.0663	0.214	0.338	0.0199
	Action + Obs.	0.453	0.368	0.1877	0.782	0.423	0.0819	0.386	0.647	0.0284
	Action + Obs. + Thou.	0.420	0.372	0.2177	0.787	0.438	0.0832	0.405	0.619	0.0218
	On-policy skill	0.304	0.219	0.1026	0.061	0.146	0.0205	0.021	0.176	0.0103
	Summarised ref.	0.349	0.218	0.0753	0.078	0.204	0.0409	0.104	0.235	0.0163
	Checklist	0.133	0.195	0.0755	−0.010	0.087	0.0042	0.000	0.196	0.0141
Self	On-policy skill	0.304	0.226	0.0937	0.121	0.216	0.0516	0.064	0.187	0.0132
	Summarised ref.	0.480	0.244	0.0714	0.093	0.160	0.0304	0.117	0.256	0.0124
Qwen3-30B-A3B-Instruct-2507										
GLM-5.2	Action	0.220	0.210	0.0627	0.727	0.208	0.0543	0.258	0.178	0.0157
	Action + Obs.	0.051	0.253	0.0813	0.757	0.238	0.0635	0.335	0.353	0.0140
	Action + Obs. + Thou.	−0.003	0.268	0.0894	0.744	0.232	0.0602	0.383	0.315	0.0112
	On-policy skill	0.211	0.210	0.0803	0.341	0.142	0.0412	0.116	0.138	0.0096
	Summarised ref.	0.179	0.222	0.0803	0.353	0.151	0.0451	0.218	0.173	0.0067
	Checklist	0.190	0.185	0.0671	0.093	0.059	0.0046	0.063	0.110	0.0093
Qwen3.7-Plus	Action	0.206	0.212	0.0704	0.670	0.207	0.0542	0.271	0.183	0.0134
	Action + Obs.	0.079	0.256	0.0829	0.683	0.233	0.0625	0.328	0.346	0.0124
	Action + Obs. + Thou.	−0.028	0.266	0.0915	0.679	0.228	0.0597	0.350	0.317	0.0111
	On-policy skill	0.116	0.157	0.0564	0.186	0.086	0.0163	0.049	0.096	0.0065
	Summarised ref.	0.050	0.153	0.0442	0.212	0.116	0.0312	0.117	0.134	0.0088
	Checklist	0.136	0.137	0.0400	0.081	0.048	0.0019	0.048	0.098	0.0071
Self	On-policy skill	0.076	0.159	0.0569	0.324	0.140	0.0398	0.059	0.115	0.0106
	Summarised ref.	0.091	0.173	0.0413	0.135	0.101	0.0277	0.193	0.128	0.0052

F PROMPTS AND EXAMPLE HINTS

F.1 AGENT PROMPTS

ALFWorld. The complete first-turn prompt for one valid_seen task:

Complete first-turn prompt, including chat tokens, for a valid_seen task.

```
<|im_start|>user
You are an expert agent solving tasks in the ALFRED household environment (TextWorld).
At each step, first reason step-by-step on a line beginning with THOUGHTS:, then give
exactly one command inside <action> </action> tags (e.g.  THOUGHTS: The fridge is closed,
so I must open it first.
<action>open fridge 1</action>).
At each turn, reason on a line beginning with THOUGHTS:, then give exactly one command
inside <action> </action> tags.  The command may be:
- look                              (describe the current location)
- inventory                         (list what you are carrying)
- go to <receptacle>                 (e.g.  go to cabinet 1)
- open <receptacle>                  (open a closed receptacle to see inside)
- close <receptacle>
- take <object> from <receptacle>    (pick up an object you can see)
- move <object> to <receptacle>      (put the object you hold onto/into a receptacle)
- use <object>                      (operate it, e.g.  use desklamp 1)
- clean <object> with <receptacle>   (e.g.  clean mug 1 with sinkbasin 1)
- heat <object> with <receptacle>    (e.g.  heat egg 1 with microwave 1)
- cool <object> with <receptacle>    (e.g.  cool apple 1 with fridge 1)
- examine <object>                  (look closely at an object or receptacle)
- slice <object> with <object>       (e.g.  slice lettuce 1 with knife 1)
Use only objects and receptacles that have appeared in the observations.
Task:  put some remotecontrol on sofa.
Observation:  -= Welcome to TextWorld, ALFRED! =-

You are in the middle of a room.  Looking quickly around you, you see a armchair 2, a
armchair 1, a coffeetable 2, a coffeetable 1, a garbagecan 1, a shelf 8, a shelf 7, a shelf
6, a shelf 5, a shelf 4, a shelf 3, a shelf 2, a shelf 1, a sofa 1, and a tvstand 1.

Your task is to:  put some remotecontrol on sofa.<|im_end|>
<|im_start|>assistant
```

AppWorld. The official AppWorld instructions and the ten-exchange demonstration precede the final user message below. A turn without a single fenced Python block receives the parse-error reply.

Final user message of the agent prompt (template variables in braces).

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

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

Parse-error reply.

````
No ```python code block found.  Emit your action as a SINGLE fenced Python block:
```python
<your code>
```
````

τ^2-bench. The 16 tool schemas are rendered into the same system block by the chat template.

Agent system prompt (the retail policy replaces the placeholder).

```
<instructions>
You are a customer service agent that helps the user according to the <policy> provided
below.
In each turn you can either:
- Send a message to the user.
- Make a tool call.
You cannot do both at the same time.

Try to be helpful and always follow the policy.  Always make sure you generate valid JSON
only.
</instructions>
<policy>
{the full retail policy wiki}
</policy>
```

Teacher context. On every benchmark, the teacher's input is the student's chat-templated prompt with the block below prepended, followed by the same sampled response. The student never sees the hint and is evaluated without it.

Teacher system block; {hint_lines} is replaced by the hint.

```
<|im_start|>system
<Secret information>
This section contains privileged hints derived from prior successful trajectories.  Use them
to stay on
track, but never mention that hints were provided.
Hint Summary:
{hint_lines}
Your Task:
1.  Follow the hints to accomplish the task still reasoning before actions.
2.  Never acknowledge having access to secret hints.
</Secret information><|im_end|>
```

F.2 Hint-Generation Prompts

F.2.1 Summarised ref.

ALFWorld and τ^2-bench use the same prompt.

Summarised ref. prompt (ALFWorld and τ^2-bench; a single user message).

```
You are creating a concise natural language hint that will help an agent solve the task.
Write a helpful hint describing the key actions and strategy the agent used.  If the agent
did not fully complete the task, focus on what they did achieve.  IMPORTANT: Output ONLY
plain text - no JSON, no code blocks, no markdown formatting.

Trajectory:
{trajectory}
```

F.2.2 On-policy skill

The analyzer prompts are benchmark-specific.

ALFWorld on-policy skill analyzer, system prompt.

```
You are an expert analyst of agent trajectories in the ALFRED/ALFWorld embodied environment
(a text world where an agent navigates a household and manipulates objects).  You read ONE
completed trajectory and its outcome and distill a TRANSFERABLE skill that would help on
similar tasks.  Never reveal or restate the exact winning action sequence verbatim; teach
the general rule, not the literal answer.  Respond with STRICT JSON only.
```

ALFWorld on-policy skill analyzer, user prompt.

```
Task: {task}
Outcome: {outcome}

Trajectory (observations and the agent's actions; <think> stripped):
{trajectory}

Return STRICT JSON with exactly these keys:
{
  "success_analysis": "what general strategy worked or would work (1-2 sentences)",
  "mistake_analysis": "the key mistake(s) and their cause, or \"\" if none (1-2
sentences)",
  "golden_workflow": ["ordered list of abstract steps (verb + target type), no literal
coordinates/answers"]
}
```

AppWorld on-policy skill analyzer, system prompt.

```
You are an expert analyst of agent trajectories in AppWorld, where an agent completes
everyday digital tasks by writing Python that calls REST-style app APIs
(`apis.<app>.<method>(...)`) across apps such as spotify, venmo, gmail, phone, simple_note,
amazon and splitwise. Key mechanics: account passwords come from the supervisor app and
each app must be logged into separately for an access token; list endpoints are paginated;
unfamiliar APIs are discovered with `apis.api_docs.show_api_descriptions` / `show_api_doc`;
and the episode only counts if the agent calls `apis.supervisor.complete_task()`. Grading
is hidden unit tests over the FINAL DATABASE STATE, so acting on too MANY entities fails
just as hard as acting on too few -- correctly scoping 'which items qualify' is usually the
crux. You read ONE completed trajectory and its outcome and distill a TRANSFERABLE skill
for similar tasks. Never restate literal values (emails, passwords, record ids, exact
answers); teach the general rule, not the specific solution. STRICT JSON only.
```

AppWorld on-policy skill analyzer, user prompt.

```
Task: {task}
Outcome: {outcome}

Trajectory (the agent's reasoning, its Python actions, and the returned output):
{trajectory}

Return STRICT JSON with exactly these keys:
{
  "success_analysis": "the general strategy that worked or would work (1-2 sentences)",
  "mistake_analysis": "the key mistake(s) and cause, or \"\" if none. Prefer naming a
SCOPING error (acted on too many/few records, wrong date window, wrong relation) or a
missing complete_task over surface details (1-2 sentences)",
  "golden_workflow": ["ordered abstract steps (verb + target type), e.g. 'resolve
credentials for the target app', 'paginate the list endpoint until exhausted', 'filter to
the qualifying subset', 'apply the mutation', 'call complete_task' -- no literal
ids/emails/answers"]
}
```

τ^2-bench on-policy skill analyzer, system prompt.

```
You are an expert analyst of agent trajectories in tau2-bench (retail/airline), where an
agent acts as a customer-service representative over a MULTI-TURN CONVERSATION with a user,
calling tools such as `find_user_id_by_email`, `get_user_details`, `get_order_details`,
`exchange_delivered_order_items`, `modify_pending_order_items` and `transfer_to_human_agents`.
Key mechanics: the agent MUST authenticate the user (email, or name + zip) before touching
any record; a written DOMAIN POLICY constrains what is permissible and the agent must refuse
or escalate when the request falls outside it; the agent must CONFIRM the details with the
user before any irreversible mutation; and it is DUAL-CONTROL -- the user holds information
and actions the agent does not, so the agent has to ELICIT what it needs rather than assume
it. Grading combines the FINAL DATABASE STATE against expected write-actions with checks
that the required information was actually communicated, so a correct mutation performed
without authenticating, confirming, or telling the user still fails. You read ONE completed
trajectory and its outcome and distill a TRANSFERABLE skill for similar tasks. Never
restate literal values (emails, user ids, order ids, item ids, prices, addresses); teach the
general rule, not the specific solution. STRICT JSON only.
```

τ^2-bench on-policy skill analyzer, user prompt.

```
Task (the user's opening request): {task}
Outcome: {outcome}

Trajectory (the conversation: agent reasoning, tool calls, user and tool responses):
{trajectory}

Return STRICT JSON with exactly these keys:
{
  "success_analysis": "the general strategy that worked or would work (1-2 sentences)",
  "mistake_analysis": "the key mistake(s) and cause, or \"\" if none. Prefer naming a
POLICY or PROTOCOL error (mutated before authenticating, skipped user confirmation, acted
outside the domain policy instead of refusing/escalating, assumed information instead of
eliciting it, wrong order/item scope) over surface details (1-2 sentences)",
  "golden_workflow": ["ordered abstract steps (verb + target type), e.g. \"authenticate
the user\", \"retrieve the account and locate the referenced order\", \"check the request
against the domain policy\", \"elicit the missing parameters from the user\", \"confirm the
exact change with the user\", \"apply the mutation\", \"report the outcome back\" -- no
literal ids/emails/prices"]
}
```

F.2.3 CHECKLIST

The three benchmarks share the checklist prompts except for the first sentence of the system prompt, which describes the environment, and ALFWorld's user prompt takes the item limit as a variable ({`max_items`}, set to 5), where the others state 5. We show the ALFWorld prompts and the other two environment sentences.

Checklist analyzer, system prompt (ALFWorld).

```
You are an expert analyst of agent trajectories in the ALFRED/ALFWorld embodied environment
(a text world where an agent navigates a household and manipulates objects). You are shown
TWO attempts at the SAME task: attempt A is by a much STRONGER REFERENCE AGENT and it
SUCCEEDED; attempt B is by the WEAKER AGENT YOU ARE ADVISING and it FAILED. Write the
PRE-ACTION CHECKLIST that would have carried B to A's outcome. The checklist is read by the
weaker agent BEFORE it acts, so every item must be an imperative it can check off in advance
-- never a retrospective observation about what happened. Teach the transferable procedure:
never name the specific entity, value, file, or answer that this particular task happens to
require, and never quote an action or line of code verbatim. An item that could only ever
apply to this one task is a FAILED item. Respond with STRICT JSON only.
```

Checklist analyzer, user prompt (ALFWorld; {`max_items`} is 5).

```
Task: {task}

Attempt A -- STRONGER REFERENCE AGENT, SUCCEEDED:
{a_trajectory}

Attempt B -- THE AGENT YOU ARE ADVISING, FAILED:
{b_trajectory}

Write at most {max_items} items per section. Return an EMPTY LIST for any section the
contrast gives you
nothing for -- do not pad, and do not restate the same instruction in more than one section.

Return STRICT JSON with exactly these keys:
{
  "before_acting": ["imperatives to satisfy before taking the first action"],
  "while_acting": ["imperatives to satisfy at every step"],
  "before_finishing": ["imperatives to satisfy before submitting the final answer"],
  "pitfalls": ["a failure mode visible in attempt B, phrased as a warning sign to watch
for"]
}
```

First sentence of the AppWorld checklist system prompt (as used).

```
You are an expert analyst of agent trajectories in the AppWorld environment (a suite of
simulated consumer apps -- email, phone, shopping, music, file system -- which the agent
drives by writing Python that calls documented `apis.<app>.<endpoint>` functions and
finishes by calling `answer(...)`).
```

First sentence of the τ^2-bench checklist system prompt.

```
You are an expert analyst of agent trajectories in the tau2-bench dual-control environment
(an agent and a simulated user collaborate over a customer-service domain, and both can act
on the shared database).
```

F.3 EXAMPLE HINTS

ALFWorld. For the training task "put a clean dishsponge in shelf" (the sponge is on shelf 1; the cleaning station is the sink basin), the hints below are shown verbatim. The *Action + Obs.* hint is the trajectory PI of the λ sweep; it opens with the reference agent's plain instruction.

Action.

```
Reference actions (in order):
1.  go to shelf 1
2.  take dishsponge 2 from shelf 1
3.  go to sinkbasin 1
4.  clean dishsponge 2 with sinkbasin 1
5.  go to shelf 1
6.  move dishsponge 2 to shelf 1
```

Summarised ref.

```
Go to shelf 1 to find a dishsponge, take it, then go to sinkbasin 1 to clean the dishsponge,
and finally return to shelf 1 to place the clean dishsponge on it.
```

On-policy skill.

```
Recommended approach:
- search shelves and surfaces for the target object
- take the target object from its current location
- go to sinkbasin
- clean the object with sinkbasin
- go to a shelf
- move the clean object to the shelf
Key skill:  The agent efficiently located the target object (dishsponge) on a shelf,
transported it to the sinkbasin for cleaning, then returned it to a shelf.  The core
strategy of find → take → clean → place worked cleanly.
```

Action + Obs. (this task has no visible pre-action prose).

```
[OBSERVATION] You are an expert agent solving tasks in the ALFRED household environment
(TextWorld).
At each step, first reason, then give exactly one command inside <action> </action> tags
(e.g.  <action>go to cabinet 1</action>).
At each turn, give exactly one command, inside <action> </action> tags.  You may use:
- look                              (describe the current location)
- inventory                         (list what you are carrying)
- go to <receptacle>                 (e.g.  go to cabinet 1)
- open <receptacle>                  (open a closed receptacle to see inside)
- close <receptacle>
- take <object> from <receptacle>    (pick up an object you can see)
- move <object> to <receptacle>      (put the object you hold onto/into a receptacle)
- use <object>                      (operate it, e.g.  use desklamp 1)
- clean <object> with <receptacle>   (e.g.  clean mug 1 with sinkbasin 1)
- heat <object> with <receptacle>    (e.g.  heat egg 1 with microwave 1)
- cool <object> with <receptacle>    (e.g.  cool apple 1 with fridge 1)
- examine <object>                  (look closely at an object or receptacle)
- slice <object> with <object>       (e.g.  slice lettuce 1 with knife 1)
Use only objects and receptacles that have appeared in the observations.
Task:  put a clean dishsponge in shelf.
Observation:  -= Welcome to TextWorld, ALFRED! =-

You are in the middle of a room.  Looking quickly around you, you see a bathtubbasin 1, a
cart 1, a garbagecan 1, a handtowelholder 2, a handtowelholder 1, a shelf 4, a shelf 3, a
shelf 2, a shelf 1, a sinkbasin 1, a toilet 1, a toiletpaperhanger 1, a towelholder 2, and a
towelholder 1.

Your task is to:  put a clean dishsponge in shelf.
[AGENT] <action>go to shelf 1</action>
[OBSERVATION] Observation:  You arrive at shelf 1.  On the shelf 1, you see a dishsponge 2,
and a soapbottle 2.
[AGENT] <action>take dishsponge 2 from shelf 1</action>
[OBSERVATION] Observation:  You pick up the dishsponge 2 from the shelf 1.
[AGENT] <action>go to sinkbasin 1</action>
[OBSERVATION] Observation:  You arrive at sinkbasin 1.  On the sinkbasin 1, you see nothing.
[AGENT] <action>clean dishsponge 2 with sinkbasin 1</action>
[OBSERVATION] Observation:  You clean the dishsponge 2 using the sinkbasin 1.
[AGENT] <action>go to shelf 1</action>
[OBSERVATION] Observation:  You arrive at shelf 1.  On the shelf 1, you see a soapbottle 2.
[AGENT] <action>move dishsponge 2 to shelf 1</action>
```

Checklist.

```
## Before you act
[ ] Systematically visit each receptacle listed in the initial observation until you find
the target object, prioritizing shelves and containers before basins
[ ] Identify which receptacle in the environment can serve as the cleaning station and which
receptacle is the final destination
[ ] Plan the full action sequence in advance:  find object, take object, navigate to
cleaning station, clean object, navigate to destination, place object

## At each step
[ ] Always navigate to a receptacle with 'go to' before attempting any interaction that
involves that receptacle, including cleaning or placing objects
[ ] If a command returns 'Nothing happens' or 'not a valid command,' immediately check
whether you are at the correct location for that action rather than repeating it
[ ] After picking up an object, explicitly navigate to the cleaning receptacle before
issuing the clean command
[ ] After cleaning, explicitly navigate to the destination receptacle before issuing the
move command

## Before you finish
[ ] Confirm that the clean command produced a success message before attempting to place the
object
[ ] Confirm that the move command produced a success message before declaring the task
complete
[ ] Do not assume the task is satisfied just because the object was found or is already on a
receptacle; every required transformation and placement must be explicitly performed and
confirmed

## Watch out for
[ ] Issuing a clean command while standing at the location where you found the object
instead of first navigating to the cleaning receptacle
[ ] Issuing a move command to a destination receptacle without first navigating to that
receptacle
[ ] Repeating the same failed command in a loop instead of diagnosing the root cause of the
failure
[ ] Concluding the task is already complete because the object was found on a receptacle,
without performing the required cleaning and placement steps
[ ] Wasting turns on 'look' or 'examine' commands that return no useful information instead
of proactively visiting receptacles to discover objects
```

AppWorld. For the training task "Follow all the classical artists on Spotify that have at least 22 followers," the three hints below are shown verbatim. The user name and e-mail address are part of AppWorld's synthetic data.

Action.

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

```python
print(apis.api_docs.show_api_doc(app_name='supervisor', api_name='show_account_passwords'))
```

```python
passwords = apis.supervisor.show_account_passwords()
spotify_password = [p for p in passwords if p['account_name'] == 'spotify'][0]['password']
login_result = apis.spotify.login(username='eri_powe@gmail.com', password=spotify_password)
print(login_result)
access_token = login_result['access_token']
```

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

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

```python
page_index = 0
all_artists = []
while True:
    result = apis.spotify.search_artists(access_token=access_token, genre='classical',
min_follower_count=22, page_index=page_index, page_limit=20)
    if not result:
        break
    all_artists.extend(result)
    page_index += 1
print(f"Found {len(all_artists)} classical artists with at least 22 followers")
for artist in all_artists:
    print(f" ID: {artist['artist_id']}, Name:  {artist['name']}, Followers:
{artist['follower_count']}")
```

```python
follow_result = apis.spotify.follow_artist(access_token=access_token, artist_id=14)
print(follow_result)
```

```python
apis.supervisor.complete_task()
```
````

On-policy skill.

```
Recommended approach:
- resolve credentials for the target app via supervisor
- login to the target app to obtain an access token
- inspect the search and mutation API docs to discover available filter parameters
- search with genre and follower-count filters, paginating until exhausted
- apply the follow mutation to every qualifying artist
- call complete_task
Key skill:  The agent leveraged the search API's built-in filters (genre and
min_follower_count) to precisely scope the qualifying artists, then paginated through all
results to ensure completeness before applying the follow mutation and calling
complete_task.
```

Checklist.

```
## Before you act
[ ] Identify every app the task requires and enumerate its available APIs before calling any
endpoint.
[ ] Determine whether the target app requires authentication, and plan to retrieve
credentials from the supervisor app and log in before making any authenticated API call.
[ ] Identify which API-level filters or parameters (e.g., genre, follower thresholds,
pagination) can directly satisfy the task criteria so you can use them instead of manual
post-filtering.

## At each step
[ ] Read the full API specification (show_api_doc) for each endpoint before calling it,
noting all required parameters and the response schema.
[ ] Pass every required parameter to each API call, including any access token obtained from
login.
[ ] Paginate through all pages of any list endpoint by incrementing the page index until an
empty page is returned, using the maximum allowed page limit.
[ ] Use API-level filters that match the task criteria directly rather than relying on a
free-text query and post-filtering results yourself.

## Before you finish
[ ] Verify that each action succeeded by inspecting the API response before moving on to the
next step.
[ ] Confirm that all pages of results were processed and no matching items on later pages
were missed.

## Watch out for
[ ] Calling APIs that require an access token without first logging in, even if results
appear to come back | the calls may not be acting on your account.
[ ] Using only a free-text query when the API supports structured filters that match the
task criteria exactly, causing you to retrieve incomplete or unfiltered results.
[ ] Stopping after the first page of results without paginating, missing matching items on
subsequent pages.
[ ] Calling an endpoint without reading its specification first, leading to omitted required
parameters such as access tokens.
```

τ^2-bench. For a training task in which the customer asks to change their default shipping address to "800 Coastal Hwy in Rehob, ZIP 19971," the GLM-5.2 hints below are shown verbatim.

Summarised ref.

```
The agent successfully changed the user's default shipping address by first authenticating
the user via email to find their user ID, then retrieving current user details to display
the existing address.  The agent clarified ambiguous information by confirming that "Rehob"
with ZIP 19971 refers to Rehoboth Beach, Delaware, and verified there was no apartment
number.  Before making any changes, the agent presented both the current and new addresses
and obtained explicit user confirmation.  Finally, the agent called the modify_user_address
function with the complete address details and confirmed the successful update to the user.
```

On-policy skill.

```
Recommended approach:
- authenticate the user via email or name plus zip
- retrieve the user account details to view the current default address
- identify any ambiguous or missing fields in the requested new address and elicit them from
the user
- present the current and proposed new address side by side and ask for explicit
confirmation
- apply the address mutation only after the user confirms
- report the updated address back to the user and offer further assistance
Key skill:  The agent followed a clean authentication-first workflow, then proactively
identified and elicited missing/ambiguous address fields (abbreviated city name, missing
state) rather than assuming, and secured explicit user confirmation with a clear
before/after comparison before mutating the record.
```

Checklist.

```
## Before you act
[ ] Identify every required field for the target modification function and check whether the
user has supplied a complete, unambiguous value for each one before presenting a
confirmation summary.
[ ] When any address component is abbreviated, partial, or missing (e.g., a shortened city
name or an absent state), proactively resolve the gap by inferring from context or asking
the user---do not pass it through as-is.

## At each step
[ ] Before calling any modification function, write down each parameter name and the exact
value you intend to assign to it, then verify the mapping is correct---city goes in the city
field, state in the state field, secondary unit in the address2 field, and so on.
[ ] If a field has no value (e.g., no apartment number), explicitly assign an empty string
to that parameter rather than reusing another value to fill it.

## Before you finish
[ ] After every modification call, inspect the returned object field-by-field and confirm
each value landed in its intended parameter---do not assume success from a non-error
response.
[ ] If the returned object shows any value in the wrong field, immediately issue a
corrective call before reporting anything to the user.

## Watch out for
[ ] Presenting a confirmation summary with missing or abbreviated fields (such as a
truncated city name and no state) shifts the burden of catching errors onto the
user---resolve ambiguities yourself first.
[ ] Swapping values between parameter fields when calling a modification function (e.g.,
placing a city name in the address-line-2 field or a state code in the city field) silently
corrupts the record while still returning a success-looking response.
```