# UNLOCKING LATENT TOOL-USE CAPABILITIES WITH PROCEDURAL HINTS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Reinforcement learning with outcome-based rewards can improve tool use, but a model may know how to execute individual actions without reliably organizing them into successful procedures. Sparse task-level feedback provides little guidance on how to correct these procedural failures. We introduce Procedural Hint Supervision (PHS), which uses reusable guidance to improve credit assignment and help models apply their existing capabilities more reliably. PHS extracts procedures by comparing successful attempts from a more capable model with failed attempts from the model being trained. During training, selected guidance re-scores the model's own trajectories, and differences between hinted and unhinted token log probabilities shape token-level advantages in two objectives built on Group Relative Policy Optimization (GRPO): PHS-Multiplicative and PHS-Additive. The trained policies are evaluated without hints. Trained separately on AppWorld and EnterpriseOps, both variants beat GRPO on held-out tasks and in transfer between the two environments. Pass@4 gains over GRPO reach 19.1 percentage points on AppWorld test (PHS-Multiplicative), 14.9 on its harder challenge split, and 7.4 on EnterpriseOps (PHS-Additive). Without benchmark training, AppWorld-trained PHS-Additive raises mean $\tau^3$ pass$^4$ across retail, airline, and telecom by 10.4 points over GRPO, and neither variant falls below the untrained model on BFCL. These results suggest that procedural guidance can unlock latent capabilities, organizing actions the model can already perform into reliable procedures that generalize across tasks, tools, and environments.

## 1 INTRODUCTION

A tool-use agent can know how to search for a record, edit it, and inspect the result, yet fail to carry out those actions in a reliable order. It may write an outcome before resolving an ambiguous identifier, trust the first page of a search, or stop without checking whether a planned change took effect. These failures do not reflect a missing atomic reasoning primitive; they reflect a failure to organize available reasoning primitives into a procedure. Improving that organization is especially important when the same procedure can be useful across applications with different tools and interfaces.

Outcome-based reinforcement learning provides a natural starting point. Group Relative Policy Optimization (GRPO) reinforces trajectories that outperform others sampled for the same task (Shao et al., 2024). This can strengthen useful procedures when successful variations are sampled. However, the final reward does not distinguish the action that prevented a failure from the many tokens surrounding it. Even where the model sometimes succeeds, learning which parts of its behavior to repeat remains a credit-assignment problem.

Privileged guidance can supply a denser signal: on-policy self-distillation (OPSD) scores a model's own rollouts with added context and trains the unprivileged policy toward that teacher (Zhao et al., 2026). Alone, such a signal can shift style rather than improve outcomes (Pan et al., 2026; Kaur et al., 2026); interactive agents still need feedback on whether a sequence of actions completes the task.

We introduce *Procedural Hint Supervision* (PHS), which combines reusable procedural guidance with outcome-based learning. We compare successful reference attempts with failed learner attempts on training tasks, extract the procedural differences, and consolidate them into a small catalog. We update the learner with GRPO, adjusting each token's advantage by how much more likely that token



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.

becomes when the task's selected procedures are added to the prompt. We study two objectives: PHS-Multiplicative, which preserves the signs of reward-derived advantages, and PHS-Additive, which also contributes credit of its own.

The central hypothesis is that the procedures provide a useful level of abstraction for transfer. Resolving an entity before acting on it or verifying a state change can remain useful when names, API signatures, and application domains/surfaces change. We therefore train separately on AppWorld and EnterpriseOps, then evaluate each policy both in its training environment and in the other as a cross-environment evaluation. We additionally evaluate BFCL function calling and $\tau^3$ customer conversations without training on either benchmark. This design tests whether in-domain improvement is accompanied by broader tool-use improvement.

Both PHS variants increase success on held-out tasks and transfer between environments: models trained on AppWorld perform better on EnterpriseOps, and vice versa. PHS-Additive also improves $\tau^3$ consistency, and neither variant falls below the untrained model on BFCL. The hint-only OPSD baseline is weaker than PHS and collapses on EnterpriseOps. On EnterpriseOps, PHS-Additive with task-matched procedures beats the same objective with reassigned or generic procedures. Prompting the untrained model with the same hints raises AppWorld test pass@4 from 40.8% to 55.7%, while PHS training reaches 78.7% with no hints at all. Together, these results suggest that procedural guidance helps models assemble actions they can already perform into complete procedures, without prescribing specific actions or teaching new ones.

## 2 From Individual Actions to Reusable Procedures

**Why a procedure can transfer.** We consider a sample workflow of verifying an entity before making a modification, such as looking up a customer in a support system, resolving a recipient in a payment application, or checking a reservation in an airline workflow. While the implementation of the workflow may differ by environment, the dependency between identification of the entity and action on the state exists across domains. A procedure in this case would describe that dependency without prescribing a task's answer. ReAct interleaves reasoning with environment actions (Yao et al., 2023), while Voyager builds and retrieves reusable behaviors during acting (Wang et al., 2023). These approaches motivate organizing behavior beyond individual actions. We ask whether reusable procedural guidance can instead supervise training, so the resulting policy carries useful behavior across tool surfaces without retrieving a procedure at deployment. This is a hypothesis about transfer, not an assumption that all failures are procedural or all capabilities are already present.

**Procedures as privileged supervision.** Learning with privileged information uses inputs available during training to improve a model that will not receive them at inference (Vapnik & Vashist, 2009). On-policy self-distillation (OPSD) makes this idea concrete for language models: a model conditioned on a reference solution supplies token-level supervision for its own unconditioned rollouts (Zhao et al., 2026). Privileged action information has also been studied for language agents (Penaloza et al., 2026). PHS builds on this scoring principle with a different source of guidance: compact procedures extracted from reference–learner disagreements are reused across training tasks. The procedure describes an action pattern while omitting the records, identifiers, and answers particular to its source task.

**What does the guidance teach?** A contextual score difference need not measure progress toward a task's outcome. Zhao et al. (2026) find that stylistic tokens can dominate full-vocabulary distillation and use pointwise divergence clipping to limit their influence. Pan et al. (2026) likewise identify style shifts in privileged-context scores and contrast correct and incorrect hints before modifying reward-derived credit. Kaur et al. (2026) report degradation on long reasoning traces associated with reduced exploration and self-correction. These findings predict that a hint signal alone, without outcome feedback, can move a policy away from task success; our hint-only baseline tests this directly.

**Guidance and verification play different roles.** A verifier measures whether a trajectory achieved the requested outcome; a procedural hint changes which tokens the model regards as appropriate along it. GRPO uses only the former, and hint-only self-distillation only the latter. PHS combines the two.

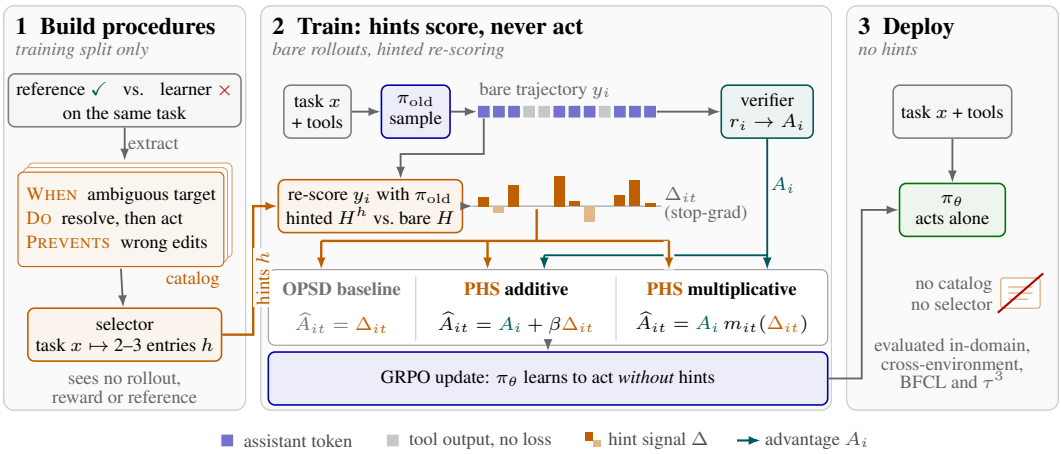


Figure 1: **Procedural Hint Supervision.** (1) A WHEN/DO/PREVENTS catalog is mined from reference/learner contrasts (entry shown is illustrative), and a selector assigns each training task 2–3 entries. (2) The frozen sampling snapshot re-scores each bare rollout with and without hints; the per-token log-probability difference $\Delta_{it}$ (stop-gradient) is combined with the verifier advantage $A_i$ in one of three objectives. (3) The trained policy is evaluated without hints.

## 3 LEARNING FROM PROCEDURAL HINT SUPERVISION

PHS mines reusable procedures offline, then uses them only to re-score the learner's own rollouts.

**Overview.** Figure 1 outlines the process from extracting the procedures to providing them as a learning signal. Mining and selection are performed on the training data only by design. During learning, selected hints produce a dense signal on the base policy's already-sampled tokens. Evaluation uses the trained policy with no procedural hinting. The catalog is therefore a source of supervision rather than an additional capability supplied to the agent at deployment.

### 3.1 MINING AND SELECTING PROCEDURES

For each training task, we compare repeated reference and learner attempts under the environment's verifier; tasks where the reference's pass rate exceeds the learner's by at least 0.2, a threshold fixed in advance rather than tuned, enter the extraction pool. An extractor contrasts a successful reference transcript with a failed learner transcript on the same task, isolating what the learner fails to organize rather than what it does not know, and states the difference as a procedure. Procedures omit task names, identifiers, values, and contact details, so they transfer across tasks and cannot leak answers; they are consolidated into one compact catalog per environment. Only training tasks are mined. Appendix A reproduces example procedures verbatim.

Each entry has WHEN, DO and PREVENTS clauses: when it applies, what to do, and which failure it prevents. For each training task, a prompted (not fine-tuned) selector picks two or three entries by procedural relevance; a few relevant entries give a sharper signal than the whole catalog. The selector sees only the task instruction and the catalog, never the learner's rollout, reward, or reference solution, so the hint depends on the task alone and cannot carry outcome information. Appendix B gives catalog sizes, task filtering, and selector inputs.

### 3.2 RE-SCORING THE LEARNER'S ROLLOUTS WITH HINTS

Let $y_i$ be a trajectory sampled without hints from the policy snapshot $\pi_{\text{old}}$, and $A_i$ its GRPO advantage: the verifier reward centered and scaled within the group of trajectories for the same task, and zero when all rewards in the group are equal. Let $H_{it}$ be the conversation prefix at assistant token $y_{it}$, and $H^h_{it}$ the same prefix, with the same tools, plus the selected procedures in the system turn. The hint signal is

$$\Delta_{it} = \text{sg}\left[\log \pi_{\text{old}}(y_{it} \mid H^h_{it}) - \log \pi_{\text{old}}(y_{it} \mid H_{it})\right]. \tag{1}$$

Both terms are teacher-forced scores of the same sampled token by the same snapshot, so re-scoring costs one forward pass and never samples a hinted trajectory; $\Delta_{it}$ is held fixed during minibatch optimization. Every trajectory is re-scored, successful or not: on a success $\Delta_{it}$ marks the tokens the hint supports, on a failure those it argues against, and scoring only one outcome would make the supervised tokens depend on reward. The policy ratio keeps the unhinted context,

$$r_{it}(\theta) = \frac{\pi_\theta(y_{it} \mid H_{it})}{\pi_{\text{old}}(y_{it} \mid H_{it})}, \tag{2}$$

so the hint changes the learning signal but not the context the policy is optimized in.

### 3.3 Multiplicative and additive objectives

The two objectives, PHS-Multiplicative and PHS-Additive, give the hint different amounts of authority. The multiplicative rule keeps the verifier as the only source of credit: the hint redistributes each trajectory's advantage across its tokens, within fixed bounds, but cannot flip its sign or create credit where the verifier gives none. A poor hint can therefore only misallocate credit, not contradict the outcome. The additive rule lets the hint contribute credit of its own. This can help when every trajectory in a group receives the same reward, so that $A_i = 0$ and GRPO has no signal, but it also lets a poor hint push against the verifier. Which trade-off suits procedural hints is not known in advance. Studying only the additive rule would not tell us whether gains depend on the hint being able to override the outcome signal.

The multiplicative rule clips the exponentiated, advantage-sign-adjusted signal, normalizes it, and blends it with uniform weighting:

$$\begin{aligned} c_{it} &= \text{clip}\big(\exp\{\text{sign}(A_i)\,\Delta_{it}\},\, 0.8,\, 1.2\big), \\ \bar{c}_i &= \frac{1}{|S_i|}\sum_{s\in S_i} c_{is}, \\ m_{it} &= 1 - \lambda + \lambda\, c_{it}/\bar{c}_i, \\ \widehat{A}^{\text{mul}}_{it} &= A_i\ \text{sg}[m_{it}]. \end{aligned} \tag{3}$$

Here $S_i$ is the implementation's normalization set, discussed below. Positive $A_i$ gives greater weight to hint-supported tokens; negative $A_i$ gives greater negative magnitude to hint-disfavored tokens. Clipping bounds how far any token's weight can move; normalization keeps a sequence's mean weight near one, so the hint redistributes rather than rescales credit (up to the tool-output effect below); and annealing $\lambda$ to zero hands credit back to the verifier, recovering the original GRPO advantages.

PHS-Additive uses the same procedural score:

$$\widehat{A}^{\text{add}}_{it} = A_i + \beta\Delta_{it}, \qquad \beta = 0.1. \tag{4}$$

We hold $\beta$ fixed without tuning and apply the correction without clipping or annealing. The formulations also differ in scale, clipping, normalization, and schedule, so their comparison tests the complete objectives rather than sign preservation alone.

For either objective, the policy component of the GRPO surrogate uses

$$\mathcal{J}(\theta) = \mathbb{E}_i\left[\frac{1}{|T_i|}\sum_{t\in T_i} \min\{r_{it}\widehat{A}_{it}, \text{clip}(r_{it}, 1-\epsilon_{\text{PPO}}, 1+\epsilon_{\text{PPO}})\widehat{A}_{it}\}\right], \tag{5}$$

where $T_i$ contains assistant tokens. This expression isolates the policy surrogate from batch reductions and any reference regularization.

### 3.4 Hint-only self-distillation baseline

Removing the outcome advantage gives our procedural-hint OPSD baseline, in which the hinted snapshot acts as a teacher for the policy's own tokens:

$$\widehat{A}^{\text{OPSD}}_{it} = \Delta_{it}. \tag{6}$$

Tokens the hinted snapshot finds more likely than the unhinted one are reinforced and less likely ones suppressed, regardless of outcome. Procedures, rollouts, scoring, and surrogate all match PHS, so the comparison isolates the verifier's contribution; only the coefficient differs (one rather than $\beta = 0.1$). The baseline adapts on-policy self-distillation (Zhao et al., 2026), in which a model conditioned on the reference solution teaches its own unconditioned rollouts through a clipped full-vocabulary divergence. Our teacher instead sees reusable procedures, and the signal is the log-ratio on sampled tokens.

**Implementation scope.** Loss is applied to assistant reasoning, tool calls, and final prose. Tool outputs are masked from loss but included in the reported multiplicative normalizer $S_i$. Their stored bare log probabilities are zero placeholders, so they also affect the scale of assistant-token advantages. Multiplicative results consequently include both token reweighting and sequence scaling. This normalization was held fixed across all reported runs; a variant normalizing over assistant tokens only was not evaluated.

## 4 Experimental Setup

**Models and environments.** We use Qwen3.6-35B-A3B as the learner, Kimi-K2.6 as the reference rollout model, and Claude Opus 4.8 as both the procedure extractor and the procedure selector.

AppWorld is a tool-use environment in which agents execute Python code to operate nine linked applications, with unit-test verification of the final state (Trivedi et al., 2024). We train on 82 tasks and evaluate on AW dev (57 tasks), AW test (168) and AW challenge (417) using the same code-ReAct harness across arms; AW and EO abbreviate AppWorld and EnterpriseOps.

EnterpriseOps evaluates tool-use workflows across eight enterprise application families, with SQL verification of the final database states. Training uses 441 tasks, and the evaluation split holds out every fifth task within the domain, yielding 128 test tasks (EO test) without filtering for hint applicability or success. The evaluation holdout contains tasks within the same application families. We use Miles (Miles Team, 2026) as the training harness, with the model interacting with EnterpriseOps environments through native tool calls.

**Training protocol.** We train AppWorld and EnterpriseOps policies separately, with no mixture of their training tasks or procedure catalogs. PHS-Multiplicative linearly anneals $\lambda$ from one to zero. PHS-Additive uses a fixed $\beta = 0.1$. The OPSD baseline uses the same procedural map with $\widehat{A} = \Delta$ and no verifier advantage. The fixed-schedule PHS-Multiplicative control holds $\lambda = 1$. Appendix B describes the configurations and training in more detail.

**Procedure access.** Catalogs and selector examples come only from training tasks. Separate diagnostics prompt the untrained model with hints on training and held-out tasks; they are not used for policy training.

**Metrics and uncertainty.** Each trained arm is one training run. In-domain and cross-environment pass@1 and pass@4 are means $\pm$ standard deviations across four seeds, each an independent evaluation run with four attempts per task. Pass@1 measures single-attempt success; pass@4 measures success within four attempts. Where a base census contains more samples, we use the finite-sample estimator in Appendix C. The selector diagnostics also report means and standard deviations over four seeds. Bold entries mark the highest displayed means, including ties, without implying statistical significance.

**External and cross-environment evaluations.** BFCL covers non-live (1,390 tasks), multi-turn (800), and live (2,251) function-calling sets (Patil et al., 2025). We evaluate retail, airline, and telecom with the $\tau^3$ implementation of the tau-bench family (Barres et al., 2025; Sierra Research, 2026b). The main external table reports BFCL pass@4 and $\tau^3$ pass$^4$, computed from four attempts per task. Pass$^4$ measures success on every attempt. Errors on these joint summaries are task-level standard errors, not SDs across seeds. The $\tau^3$ mean gives equal weight to its three domains. The appendix additionally reports $\tau^3$ pass@4, single-attempt domain success, and single-attempt BFCL category accuracy.

Cross-environment evaluation applies each checkpoint to the other environment's harness without hints or target-environment training.

## 5 Task Success and Generalization

Policies act without hints; OPSD is the hint-only baseline. Blue/orange: above/below GRPO; grey: seed SDs, or task-level standard errors for $\tau^3$ and BFCL; bold: highest mean. AW/EO: AppWorld/EnterpriseOps.

### 5.1 Held-out success

We first ask whether PHS helps on the tasks it was trained for. Each policy is evaluated on unseen tasks from its own environment: AW dev, test and challenge for AppWorld, and EO test for EnterpriseOps. Pass@4 counts a task as solved if any of four attempts succeeds.

*Both PHS variants beat GRPO on every held-out split, by 19.1 pass@4 points on AW test (PHS-Multiplicative) and 7.4 on EO test (PHS-Additive); OPSD falls below GRPO on three of four splits and below the base on EO test.*

Table 1: Held-out success, pass@4 (%)

| Evaluation | Base | GRPO | OPSD | PHS-Multiplicative | PHS-Additive |
|---|---|---|---|---|---|
| AW dev | 32.89 ± 0.88 | 48.68 ± 5.97 | 40.35 ± 3.79 | **67.98** ± 2.21 | 64.04 ± 3.04 |
| AW test | 40.77 ± 2.94 | 59.67 ± 1.97 | 60.12 ± 1.75 | **78.72** ± 1.71 | 74.40 ± 1.09 |
| AW challenge | 16.85 ± 1.11 | 34.17 ± 0.14 | 25.96 ± 0.84 | **49.10** ± 0.66 | 45.92 ± 1.15 |
| EO test | 26.59 ± 0.96 | 33.01 ± 0.39 | 4.88 ± 0.75 | 37.50 ± 1.42 | **40.43** ± 1.73 |

Mean ± SD over four seeds. AW dev, test and challenge have 57, 168 and 417 tasks; EO test has 128. Policies are trained separately per environment.

OPSD's collapse on EO test is the failure the motivation predicts for a hint signal without outcome feedback.

### 5.2 Transfer

Transfer asks whether what PHS teaches is specific to one environment. Each policy runs in the other environment's harness, with different applications, tools and interfaces, and with no training or hints there. Gains that came only from environment-specific behavior would vanish here.

*Both PHS variants beat GRPO on every transfer split; PHS-Multiplicative gains 6.1 pass@4 points over GRPO from AppWorld to EO test and 12.5 from EnterpriseOps to AW test. GRPO does not improve on the untrained base on any transfer split.*

Table 2: Cross-environment transfer, pass@4 (%)

| | AppWorld → | EnterpriseOps → | | |
|---|---|---|---|---|
| Arm | EO test | AW dev | AW test | AW challenge |
| Base | 26.59 ± 0.96 | 32.89 ± 0.88 | 40.77 ± 2.94 | 16.85 ± 1.11 |
| GRPO | 25.00 ± 0.64 | 26.32 ± 0.00 | 37.80 ± 3.72 | 15.89 ± 0.41 |
| OPSD | 28.71 ± 2.05 | 27.19 ± 1.01 | 43.90 ± 1.32 | 18.21 ± 0.67 |
| PHS-Multiplicative | **31.06** ± 2.42 | 37.72 ± 4.64 | **50.30** ± 1.72 | **23.38** ± 0.99 |
| PHS-Additive | 30.28 ± 0.75 | **38.16** ± 1.68 | 48.81 ± 0.00 | 21.76 ± 0.79 |

Mean ± SD over four seeds; no target-environment training or hints. EnterpriseOps has a single held-out split.

OPSD lands between GRPO and PHS on every split.

### 5.3 $\tau^3$ consistency

$\tau^3$ tests customer-service conversations, which neither training environment contains. We report pass$^4$, the share of tasks solved in all four attempts, because consistency is where a missing procedural step shows most: a policy that sometimes skips verifying a change fails the task.

*PHS-Additive beats GRPO in all six domain–training pairs, while EnterpriseOps-trained OPSD falls below GRPO in all three domains.*

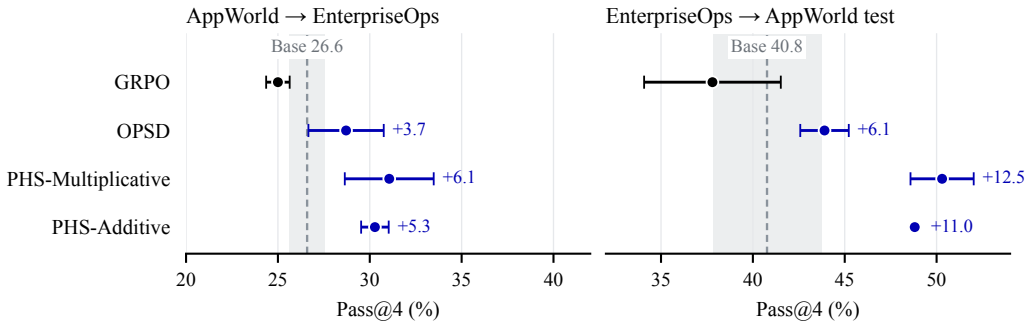


Figure 2: Cross-environment transfer, pass@4 (%).

Labels give the change from GRPO; the grey band is the untrained base, mean ± SD.

Table 3: $\tau^3$ consistency, pass$^4$ (%)

| | AppWorld-trained | | | EnterpriseOps-trained | | |
|---|---|---|---|---|---|---|
| Arm | Retail | Airline | Telecom | Retail | Airline | Telecom |
| Base | **60.53** ± 4.58 | 54.00 ± 7.05 | 56.14 ± 4.65 | **60.53** ± 4.58 | 54.00 ± 7.05 | 56.14 ± 4.65 |
| GRPO | 54.39 ± 4.66 | 60.00 ± 6.93 | 60.53 ± 4.58 | 57.89 ± 4.62 | 62.00 ± 6.86 | 54.39 ± 4.66 |
| OPSD | **65.79** ± 4.44 | **62.00** ± 6.86 | 53.51 ± 4.67 | 44.74 ± 4.66 | 14.00 ± 4.91 | 42.98 ± 4.64 |
| PHS-Multiplicative | **66.67** ± 4.41 | 60.00 ± 6.93 | 60.53 ± 4.58 | **60.53** ± 4.58 | **72.00** ± 6.35 | **60.53** ± 4.58 |
| PHS-Additive | **70.18** ± 4.28 | **72.00** ± 6.35 | **64.04** ± 4.49 | **65.79** ± 4.44 | 64.00 ± 6.79 | **66.67** ± 4.41 |

Tasks solved in all four attempts, ± task-level standard error. No $\tau^3$ training or evaluation hints.

Weighting domains equally, AppWorld-trained PHS-Additive reaches 68.74% against 58.31% for GRPO. PHS-Multiplicative ties GRPO on two AppWorld-trained domains, and PHS-Additive's gain exceeds two combined standard errors only on AppWorld-trained retail.

### 5.4 BFCL FUNCTION CALLING

BFCL tests single-call and multi-turn function calling without environment state or long task horizons. It therefore measures whether PHS training costs general function-calling ability, rather than whether it improves agentic procedures.

*BFCL barely moves: no trained arm differs from the untrained base by more than two standard errors, and PHS never falls below it.*

Table 4: BFCL function calling, pass@4 (%)

| | AppWorld-trained | | | EnterpriseOps-trained | | |
|---|---|---|---|---|---|---|
| Arm | Non-live | Multi-turn | Live AST | Non-live | Multi-turn | Live AST |
| Base | **86.02** ± 1.15 | 70.62 ± 1.59 | 83.12 ± 1.02 | **86.02** ± 1.15 | 70.62 ± 1.59 | **83.12** ± 1.02 |
| GRPO | 84.46 ± 1.21 | 72.25 ± 1.57 | 83.35 ± 1.01 | 83.92 ± 1.23 | 70.75 ± 1.60 | 83.05 ± 1.02 |
| OPSD | **86.29** ± 1.14 | 71.50 ± 1.58 | 82.90 ± 1.02 | **88.15** ± 1.07 | **75.25** ± 1.51 | **83.64** ± 1.01 |
| PHS-Multiplicative | **86.02** ± 1.15 | **72.38** ± 1.56 | **83.79** ± 1.00 | **86.48** ± 1.14 | **74.88** ± 1.52 | **83.79** ± 1.00 |
| PHS-Additive | **86.58** ± 1.13 | 71.00 ± 1.58 | **83.64** ± 1.01 | **86.83** ± 1.12 | **73.00** ± 1.55 | **83.64** ± 1.01 |

Four attempts per task, ± task-level standard error. No BFCL training or evaluation hints.

### 5.5 ABLATIONS

We test two parts of PHS, changing one and keeping the rest fixed (OPSD also changes the hint coefficient; see below): the combination of reward and hint, and the selection of procedures. A third, annealing $\lambda$ in PHS-Multiplicative rather than holding it at one, changes pass@4 by at most 1.7 points on any split (Appendix D).

#### 5.5.1 REWARD AND HINT CONTENT

We run these controls with PHS-Additive on EnterpriseOps. It is the stronger variant there, and its hint term still gives a learning signal when every trajectory in a group receives the same reward

($A_i = 0$), where the multiplicative weight gives none; removing the reward term therefore leaves a trainable objective. Removing the hint term from PHS-Additive gives GRPO; removing the reward term gives OPSD, which also raises the hint coefficient from 0.1 to 1. We then keep PHS-Additive fixed and replace each task's procedures with those selected for another task, or with one generic procedure.

*On EnterpriseOps, PHS-Additive needs both the outcome reward and procedures matched to the task.*

Table 5: Reward and procedure controls, EnterpriseOps pass@4 (%)

| Objective | Pass@4 (%) |
|---|---|
| GRPO (reward only) | $33.01 \pm 0.39$ |
| OPSD (hint only) | $4.88 \pm 0.75$ |
| PHS-Additive | $\mathbf{40.43} \pm 1.73$ |
| PHS-Additive, reassigned procedures | $33.01 \pm 2.33$ |
| PHS-Additive, one generic procedure | $29.88 \pm 0.75$ |

Mean ± SD over four seeds.

Removing either term loses the gain: OPSD falls below the untrained base, and GRPO is 7.4 points below PHS-Additive. Reassigned procedures match GRPO and a generic procedure falls below it, consistent with the gain coming from task-matched hints rather than added text.

### 5.5.2 Procedure selection

PHS's training hints come from a selector (Claude Opus 4.8) prompted with the task, the catalogue and a few training tasks paired with their mined procedures. We test this step alone, with no policy training: we prompt the untrained learner (Qwen3.6-35B-A3B) with procedures chosen at random, by the selector with or without those examples, or by the learner itself using the same prompt with examples.

*Random procedures help on AW test and challenge; selected procedures help on every split.*

Table 6: Prompting the untrained model with hints, pass@4 (%)

| Hints | AW dev | AW test | AW challenge | EO test |
|---|---|---|---|---|
| No hint | $32.89 \pm 0.88$ | $40.77 \pm 2.94$ | $16.85 \pm 1.11$ | $33.98 \pm 1.01$ |
| Random procedures | $34.21 \pm 2.26$ | $48.51 \pm 1.79$ | $21.70 \pm 0.60$ | $35.55 \pm 2.43$ |
| Selector, no examples | $36.40 \pm 3.00$ | $52.38 \pm 1.46$ | $22.72 \pm 0.77$ | $35.94 \pm 1.91$ |
| Selector with training examples (PHS) | $41.67 \pm 3.89$ | $\mathbf{55.65} \pm 3.42$ | $23.80 \pm 0.53$ | $39.45 \pm 1.01$ |
| Learner as selector | $\mathbf{44.30} \pm 4.83$ | $52.98 \pm 1.46$ | $\mathbf{24.10} \pm 1.05$ | $\mathbf{41.60} \pm 1.73$ |

Blue: above the no-hint prompt. No policy training; mean ± SD over four seeds; separate campaign from Table 1 (EO base 26.59 there).

The selector with examples beats random procedures on every split, and the learner prompted as its own selector has the higher mean on three of four splits and is within one SD on the fourth. This tests the quality of the hints, not whether a better selector trains a better policy.

*Training moves the procedures into the weights: PHS-Multiplicative, tested with no hints, beats the untrained model given hints in its prompt by more than 23 pass@4 points on every AppWorld split.*

Table 7: Hints at prompt time versus PHS training, AppWorld pass@4 (%)

| Model | Hints at test | AW dev | AW test | AW challenge |
|---|---|---|---|---|
| Untrained | none | $32.89 \pm 0.88$ | $40.77 \pm 2.94$ | $16.85 \pm 1.11$ |
| Untrained | in prompt | $41.67 \pm 3.89$ | $55.65 \pm 3.42$ | $23.80 \pm 0.53$ |
| PHS-Multiplicative | none | $\mathbf{67.98} \pm 2.21$ | $\mathbf{78.72} \pm 1.71$ | $\mathbf{49.10} \pm 0.66$ |
| PHS-Additive | none | $64.04 \pm 3.04$ | $74.40 \pm 1.09$ | $45.92 \pm 1.15$ |

PHS rows use hints only during training. Mean ± SD over four seeds; blue: above the untrained model without hints.

The prompted gains show that the hints are informative; the trained gains show that PHS makes the same behavior available without them.

## 6 DISCUSSION

**Hints combined with reward are a robust learning signal.** The gains come from pairing procedural hints with the outcome reward, not from a particular way of combining them. On every held-out and transfer split, the gap between PHS-Multiplicative and PHS-Additive is at most 4.3 pass@4 points, smaller than either variant's gain over GRPO, which is at least 4.5. Both variants also beat hint-only OPSD on all eight splits. The result is robust to how strongly and when the hint is applied: holding the hint weight fixed instead of annealing it changes pass@4 by at most 1.7 points. It is also robust to how hints are chosen: random procedures, a selector with or without examples, and the learner selecting its own hints all raise the untrained model's success. What matters is the content: procedures matched to the task, not reassigned or generic ones.

**Reward and hint play different roles.** Removing either one loses the gain, because each signal alone has a flaw the other fixes. GRPO gives every token of a trajectory the same credit, so it cannot tell the step that prevented a failure from the tokens around it; its in-domain gains do not carry over, and it falls below the untrained model in transfer. Hint-only OPSD assigns dense credit but has no notion of whether the task succeeded; it collapses on EnterpriseOps while scoring highest on BFCL, so strong function-calling scores alone do not indicate a more capable agent. The reward decides whether a trajectory succeeded; the hint decides which of its steps deserve the credit. The two variants differ only in how much authority the hint has. PHS-Multiplicative, which leaves the verifier in charge of credit, leads on AppWorld, and PHS-Additive, which lets the hint add credit of its own, leads on EnterpriseOps and $\tau^3$; either choice beats both signals used alone.

**The capability is largely latent.** If PHS taught new abilities, hints given at prompt time should help little. Instead, prompting the untrained model with the selected procedures already lifts it on every AppWorld split, and PHS training then goes much further with no hints at all. The procedures therefore appear to organize actions the model can already perform rather than teach new ones, and training moves them into the weights.

**A training signal that scales.** PHS adds little compute or data on top of GRPO. Procedures are mined once, from a small number of contrasts (25 pairs give the 12 AppWorld procedures; Appendix A), and re-scoring costs one teacher-forced forward pass per trajectory with no extra sampling. The learner selects useful hints nearly as well as a stronger model, so selection need not depend on a stronger model. And because the hint only reshapes credit inside the reward's verdict, it inherits the grounding of the verifier rather than the drift of distillation alone. These properties make procedure-guided credit assignment a practical signal for larger models and more environments, which we pursue next.

**Why procedures transfer.** Resolving a reference before acting, checking a precondition before changing a record, and completing every part of a request are useful across applications with different tools. For example, the EnterpriseOps procedure *Execute every required mutation*, selected for 247 of 441 training tasks, checks that every requested change is performed; BFCL's parallel category fails the same way when a request needs several calls and only one is made, and EnterpriseOps-trained PHS raises its accuracy from 73.00% under GRPO to over 83%. Appendix A gives the procedure texts, and Appendix E shows where the same behaviors recur in external tasks. Which learned procedure produces a given gain remains open, since we measure task success rather than individual procedures.

## 7 CONCLUSION

Procedural Hint Supervision uses procedures mined from strong-versus-weak contrasts to assign token-level credit during training, with no hints at test time. Combined with the outcome reward, the hint improves held-out success and transfer between tool-use environments under either combination rule, either schedule and any selector we tried, and helps on external benchmarks without training on them. Pairing guidance about how to act with feedback on whether the task succeeded is a simple and robust paradigm for turning actions a model can already perform into reliable procedures.

## AI USE STATEMENT

Generative models supplied reference trajectories, extracted procedural guidance, and selected catalog entries as described in the method. AI assistance was also used for literature lookup, experimental design feedback, interpretation of results, mathematical exposition, manuscript drafting and organization, numerical transcription, and preparation of tables and plotting code. The authors are responsible for verifying the source data, analysis, citations, and final claims.

## REPRODUCIBILITY STATEMENT

Section 3 describes the training objectives and hint-scoring procedure. Appendix B documents the data splits and hint construction, Appendix B.5 gives the training hyperparameters, and Appendix C defines the evaluation metrics.

## REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. $\tau^2$-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025. URL https://arxiv.org/abs/2506.07982.

Gorilla Team. BFCL V3: Multi-turn and multi-step evaluation, 2024. URL https://gorilla.cs.berkeley.edu/blogs/13_bfcl_v3_multi_turn.html.

Gorilla Team. BFCL: Public task definitions and evaluation code. Official repository, pinned example snapshot, 2026. Example snapshot at commit 6ea57973c7a6, accessed September 23, 2026.

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. URL https://arxiv.org/abs/2607.05184.

Miles Team. Miles: Enterprise-grade reinforcement learning for large-scale model post-training. https://github.com/radixark/miles, 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. URL https://arxiv.org/abs/2606.11709.

Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://gorilla.cs.berkeley.edu/leaderboard.

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. URL https://arxiv.org/abs/2602.04942.

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. URL https://arxiv.org/abs/2402.03300.

Sierra Research. $\tau^3$-bench: Public task definitions and domain policies. Official repository, pinned example snapshot, 2026a. Example snapshot at commit b7ea9074c1cb, accessed September 23, 2026.

Sierra Research. $\tau^3$-bench: Official implementation and release documentation, 2026b. URL https://github.com/sierra-research/tau2-bench. Accessed September 11, 2026.

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. URL https://aclanthology.org/2024.acl-long.850/.

Vladimir Vapnik and Akshay Vashist. A new learning paradigm: Learning using privileged information. *Neural Networks*, 2009. doi: 10.1016/j.neunet.2009.06.042. URL https://www.sciencedirect.com/science/article/pii/S0893608009001130.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. URL https://arxiv.org/abs/2305.16291.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

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. URL https://arxiv.org/abs/2601.18734.

# A EXAMPLES FROM THE PROCEDURE LIBRARIES

The entries below reproduce the WHEN, DO, and PREVENTS clauses verbatim. The EnterpriseOps catalog contains 20 procedures extracted from 71 reference–learner disagreements; the AppWorld catalog contains 12, extracted from 25. For each training task, the selector concatenates three AppWorld entries or two to three EnterpriseOps entries under a one-line header. This text enters only the scoring context after a trajectory has been sampled.

Counts give the number of distinct training tasks whose map selects the procedure. A task can select several entries, so these counts overlap. They describe catalog use across tasks, rather than token-level influence.

## A.1 ENTERPRISEOPS: COMPLETING ACTIONS AND RESOLVING DEPENDENCIES

**Execute every required mutation**
*Selected for 247 of 441 training tasks.*

**WHEN** A task states multiple distinct end-states or side-effects, each needing its own write, after shared lookups.

**DO** Decompose the request into every required state change; issue a separate write for each rather than assuming one update cascades them; verify each verifiable end-state has a corresponding successful write.

**PREVENTS** Partial completion where only some required mutations run, leaving records in prior states or associations unmade.

**Thread created ids into follow-ups**
*Selected for 118 of 441 training tasks.*

**WHEN** A newly created entity must be added to a container or referenced by later mutations.

**DO** Capture the id from the create response, resolve any target container ids, then pass those exact captured ids into each downstream membership-add or update call, verifying each write echoes the intended entity.

**PREVENTS** Downstream writes silently targeting a wrong or unresolved id despite the tools being invoked.

**Resolve references before writing**
*Selected for 114 of 441 training tasks.*

**WHEN** A create or update must reference entities named by human labels, or that depend on parent records, before the write.

**DO** Call lookup/list tools for every referenced entity in dependency order, confirm each returned the intended record, then pass the exact resolved ids into the mutation.

**PREVENTS** Writes rejected or silently targeting nothing because a required reference was unresolved, mismatched, or wrong-scope.

**Confirm gating condition before acting**
*Selected for 112 of 441 training tasks.*

**WHEN** Destructive or creation steps are gated on an entitlement, knowledge, or data condition verified against a source record.

**DO** Look up the governing record, evaluate its state against the condition, and only proceed with the mutations if it holds; branch to the specified alternative otherwise.

**PREVENTS** Acting on an invalid or wrong entity because the required precondition check was skipped.

## A.2 ENTERPRISEOPS: SELECTING VALUES AND CHECKING COMPLETION

**Normalize values to field format**
*Selected for 87 of 441 training tasks.*

**WHEN** A supplied value is ambiguous, relative, or malformed and must be persisted, or a literal payload must be saved verbatim.

**DO** Interpret the intended canonical value, convert relative dates to absolute timestamps and locale-formatted numbers to typed values, or pass verbatim structures unchanged; set all required flags explicitly, then read back to confirm.

**PREVENTS** Persisting wrong, misparsed, or altered values so the state verifier for the mutated record fails.

**Select candidate matching task subject**
*Selected for 82 of 441 training tasks.*

**WHEN** A search returns multiple candidates and one must be linked to a target.

**DO** Filter results by the specific descriptors named in the task (product, version, topic, device), select the record that actually matches, capture its concrete id, then perform the link before any dependent notification.

**PREVENTS** Linking the wrong or no record because the retrieval result wasn't matched to the task's stated subject.

**Continue through entire task chain**
*Selected for 75 of 441 training tasks.*

**WHEN** A multi-step task chains dependent creation, membership, messaging, and notification actions.

**DO** Enumerate all required actions up front, execute each in dependency order verifying preconditions and persistence, and continue past the first action through the final notification or verification step.

**PREVENTS** Abandoning the task after early actions, leaving downstream state changes and the closing deliverable unperformed.

**Confirm phase transition persisted**
*Selected for 67 of 441 training tasks.*

**WHEN** Updating a workflow record that must both change fields and advance to a new phase or state.

**DO** Issue the update with the explicit next-phase value alongside field changes, then re-read the record to confirm both the field writes and the state progression actually persisted.

**PREVENTS** Leaving the record in its original phase because field updates succeeded but the required state transition was omitted.

### A.3 APPWORLD: COMPLETE RETRIEVAL, VERIFICATION, AND ENDPOINT CHOICE

**Exhaustively paginate every list endpoint before deriving conclusions**
*Selected for 55 of 82 training tasks.*

**WHEN** Any endpoint supports page_index/page_limit and the task needs 'all' matching items or an aggregate.

**DO**

1. Set page_limit to its documented maximum.
2. Loop incrementing page_index, accumulating results until a page returns empty or fewer items than the limit.
3. Add a safety cap to avoid infinite loops and print the total count.
4. Aggregate/filter over the complete set; never hardcode IDs from a single visible page.

**PREVENTS** Silently dropping items beyond the first page, corrupting intersections/extrema and failing completeness checks.

**Verify the result and finalize with complete_task in the expected format**
*Selected for 38 of 82 training tasks.*

**WHEN** You have executed the mutating action(s) or computed a confident answer and are about to finish.

**DO**

1. Capture and print each action's response to confirm the intended effect; re-fetch affected collections to verify the end state.
2. Execute any explicit terminal instruction (e.g. delete/terminate) only after verifying prerequisite artifacts.
3. Call apis.supervisor.complete_task, passing only the concise requested value or no answer if none is requested.

**PREVENTS** Declaring completion on a silently-failed action, leaving the task unsignaled, or failing unit tests on answer-format mismatch.

**Match task vocabulary to the exact dedicated endpoint instead of a plausible proxy**
*Selected for 11 of 82 training tasks.*

**WHEN** A task uses a specific noun or qualifier (library, liked, recommended, social feed, inbox, request, following) and multiple similarly-named APIs exist.

**DO**

1. Scan the full API list for a name/description that literally matches the task's term.
2. Prefer that direct-match endpoint over a semantically-similar one or a generic search plus manual filtering.
3. Confirm via its response schema that it returns the exact entity/attribute the task needs.
4. Fall back to an approximate source only if no exact-match API exists, and note the assumption.

**PREVENTS** Operating on the wrong dataset (e.g. liked songs vs song library, transactions vs social feed) so results look plausible but fail unit tests.

# B Protocol, Data Access, and Reproducibility

## B.1 Dataset construction and split accounting

AppWorld uses 82 training tasks. Procedure extraction draws on 25 discordant reference–learner pairs and yields 12 consolidated procedures. AW dev (57 tasks), AW test (168) and AW challenge (417) are excluded from mining and training. Challenge evaluation takes place after the checkpoints are frozen.

EnterpriseOps uses 441 training tasks and 128 held-out tasks. The training pool excludes 76 tasks with zero observed reward variance: 68 never rewarded and eight always solved. This filtering applies only to training. The holdout retains 76 tasks never fully solved by the base census, 51 sometimes solved, and one always solved.

The base census uses four seeds and eight samples per task. Its train and test pass@4 estimates are approximately 0.271 and 0.266. A domain-stratified task split limits some forms of selection bias but does not imply disjoint procedural situations, application APIs, or task templates.

The training-data filter identifies observed reward variation, not a fundamental capability boundary. A task with no reward in the finite census may remain recoverable with guidance or additional sampling. The mining criterion similarly targets reference–learner disagreements rather than tasks the learner already solves on every attempt. These choices concern which tasks provide training or extraction data; they are distinct from the scoring procedure applied to each retained training trajectory.

## B.2 Model roles and hint provenance

Table 8: Model roles in the pipeline

| Role | Specification |
|---|---|
| Learner | Qwen3.6-35B-A3B. |
| Reference rollout model | Kimi-K2.6. |
| Procedure extractor | Claude Opus 4.8; extracts reusable procedures from successful and failed training transcripts. |
| Task selector | Claude Opus 4.8 with catalog constraints and training-task examples. |
| Token scorer | Frozen learner sampling snapshot under bare and hinted contexts. |
| Task verifier | Environment unit tests or SQL checks over final state. |

The stronger reference model supplies training transcripts only; it is not the token-scoring teacher.

The EnterpriseOps selector receives at most 2,000 task-instruction characters, the 20-entry catalog, and few-shot training-task/procedure examples, with a budget of three entries. Alignment is in prompt examples and rules. Multiple procedures can legitimately fit one task. The selector receives no learner rollout, reward, or reference solution. A separate verifier-targeted map using failing conditions and tool calls was not consumed by the training arms.

The training maps contain nonempty hints for all 82 AppWorld and 441 EnterpriseOps training tasks. Each AppWorld entry contains exactly three procedures; EnterpriseOps entries contain two or

three. The harness supports an empty entry by training that task with plain GRPO, but no consumed map entry is empty. Every trajectory in hint-enabled training receives the additional teacher-forced scoring query, regardless of its reward. Hint strength follows the specified objective and training schedule, not an online decision to intervene on a failed rollout.

### B.3 Prompt and catalog construction

The pipeline separates four uses of procedural information:

- **Extraction:** compare paired training transcripts to identify a reusable procedural difference; exclude task names, IDs, concrete values, email addresses, and contacts; emit applicability, action, and failure clauses.
- **Consolidation:** deduplicate the extracted procedures while preserving their source-pair provenance.
- **Selection:** choose existing catalog IDs from a task instruction and training-only examples, subject to the entry budget and procedural applicability.
- **Scoring:** insert the selected block into the system turn, preserving the observed conversation and sampled response tokens.

### B.4 Prompt and hint lengths

Table 9: System-prompt and selected-hint lengths

| Population | System tokens mean (median) | Hint tokens mean | Reported ratio (%) | Median ratio (%) | Ratio range (%) |
|---|---|---|---|---|---|
| AppWorld train-82 | 311 (—) | 461 | 148[a] | 150 | 129–170 |
| EnterpriseOps train-441 | 1,913 (2,078) | 288 | 15[b] | 13.7 | 5–1,600 |
| EnterpriseOps test-128 | 1,860 (2,108) | 290 | 15.6[b] | 13.7 | — |

[a]Mean of per-task hint/system ratios; [b]ratio of mean hint length to mean system-prompt length. Medians and ranges are per task; a dash marks an unavailable statistic. The EO test row describes diagnostic hint text, not hints used in training or the main evaluations. Ratios compare hints with the system prompt, not the full conversation.

Selected hint blocks average 461 tokens on AppWorld training tasks and 288 on EnterpriseOps training tasks (Table 9). They extend only the scoring context; no hint tokens are added to the learner's sampled episodes. The AppWorld system prompt is a short harness protocol and task instruction, with the instruction accounting for 29 of its 311 mean tokens. A hint is therefore approximately one and a half times that system prompt. EnterpriseOps has longer application operating prompts, giving a hint/system ratio of approximately 15% by ratio of means.

Per-task ratios and ratios of means answer different questions. Very short EnterpriseOps system prompts, some under 100 characters, produce large per-task ratios; the reported training range reaches 1,600%. We report median ratios and ratios of means separately. These measurements do not establish the relative cost of re-scoring complete conversations. The held-out measurements describe diagnostic hint text only and do not change the training and evaluation access rules.

### B.5 Training configuration

We fine-tune all parameters of Qwen3.6-35B-A3B using fully sharded data parallelism (FSDP) in bfloat16. Table 10 gives the shared optimization and sampling settings. Each rollout batch forms one optimization minibatch, with one optimizer step per batch; token-based packing uses a budget of 9,216 tokens per GPU. Reward-based objectives use group-normalized advantages from the verifier's partial reward. OPSD instead uses the procedural score signal alone.

Table 10: Shared optimization and sampling settings

| Setting | Value |
|---|---|
| Learner | Qwen3.6-35B-A3B; full-parameter fine-tuning |
| Precision and parallelism | bfloat16; FSDP; gradient checkpointing enabled |
| Optimizer | Adam; $\beta_1 = 0.9$, $\beta_2 = 0.98$; weight decay 0.1 |
| Learning rate | $10^{-6}$, constant throughout training |
| PPO ratio clipping | Lower $\epsilon = 0.2$, upper $\epsilon = 0.28$; ratio bounds $[0.8, 1.28]$ |
| KL and entropy coefficients | Both zero |
| Sampling temperature | 1.0 |
| Hardware layout | One node with eight GPUs |
| Rollout engines | Eight SGLang engines, one per GPU, colocated with training |

Table 11 gives the environment-specific batch and episode limits. AppWorld uses batches of 16 tasks with eight samples each; EnterpriseOps uses eight tasks with eight samples each and Miles as the training harness. Selected procedures are appended to the system turn only for the additional scoring pass and never enter the sampling context.

Table 11: Environment-specific training settings

| Setting | AppWorld | EnterpriseOps |
|---|---|---|
| Training tasks | 82 | 441 |
| Tasks per rollout batch | 16 | 8 |
| Samples per task | 8 | 8 |
| Trajectories per optimization minibatch | 128 | 64 |
| Maximum turns per episode | 30 | 50 |
| Response token cap | 61,440 | 24,576 |
| Context token limit | 65,536 | 131,072 |

AppWorld additionally caps each turn at 3,000 tokens. EnterpriseOps runs 16 environments concurrently.

Table 12: Objective-specific training settings

| Training condition | Configuration |
|---|---|
| GRPO | Bare sampling and reward-derived advantages |
| PHS-Multiplicative | Linear $\lambda : 1 \to 0$; raw token weights clipped to $[0.8, 1.2]$ before normalization |
| PHS-Additive | Same procedural score signal; fixed $\beta = 0.1$; no hint-signal clipping |
| OPSD | Same procedural score signal alone; $\widehat{A} = \Delta$, coefficient one |
| PHS-Additive, reassigned procedures | Same catalog; reassigned task-to-hint mapping |
| PHS-Additive, one generic procedure | One shared generic procedure |
| PHS-Multiplicative, fixed $\lambda$ | $\lambda = 1$ throughout; same token-weight clipping |

## C Metric Definitions and Uncertainty

For task $j$, let $s_{jr} \in \{0, 1\}$ denote full success on attempt $r$ and $n_j$ the number of attempts. The direct per-task pass@1 estimate is $\sum_r s_{jr}/n_j$, averaged over tasks. With $c_j$ successes among $n_j \geq k$ independent exchangeable samples, the usual finite-sample pass@k estimator is

$$\widehat{\text{pass@}k}_j = 1 - \frac{\binom{n_j - c_j}{k}}{\binom{n_j}{k}}. \tag{7}$$

For exactly four attempts and $k = 4$, this reduces to the indicator that any attempt succeeded. The all-four consistency metric is

$$\widehat{\text{pass}^4} = \frac{1}{N} \sum_{j=1}^{N} \prod_{r=1}^{4} s_{jr}. \tag{8}$$

For in-domain and cross-environment evaluations, pass@1 and pass@4 are summarized as means and SDs over four seeds. The base EnterpriseOps census uses eight samples per task per seed and the finite-sample estimator above. For the external benchmarks, four attempts are combined into one pass@4 or pass$^4$ score per task set. Their displayed errors are task-level standard errors, not seed SDs. The $\tau^3$ single-attempt success tables instead report means and SDs over four seeds. Retail, airline, and telecom have 114, 50, and 114 tasks; their reported macro mean weights domains equally. BFCL pass@4 is computed from per-task outcomes and category aggregation, not inferred from mean accuracy. Non-live and multi-turn each weight four categories equally; the non-live simple category averages its Python, Java, and JavaScript parts. Live weights its four AST-scored categories by task count and excludes relevance and irrelevance items from this aggregate.

The frozen-base selector diagnostics also use four seeds. Their EnterpriseOps no-hint baseline is 33.98% pass@4, compared with 26.59% in the main census. These are separate evaluation campaigns and are not treated as interchangeable baselines. The selector results support comparisons within that prompting campaign; differences across campaigns are not interpreted as training effects.

The plotted error bars are marginal SDs. They are not paired confidence intervals, significance tests, or evidence of non-inferiority.

## D Annealed versus Fixed Hint Weighting

This ablation tests whether the multiplicative method benefits from reducing procedural influence over training. The main multiplicative experiments linearly anneal $\lambda$ from one to zero, gradually returning to the unmodified GRPO advantage. The fixed condition holds $\lambda = 1$ throughout. Both use the same selected procedures and the same clipped, normalized weighting rule; $\lambda$ determines how strongly those weights modify the outcome-derived advantage. Table 13 compares the schedules on both training environments.

Table 13: Hint-weight schedules for PHS-Multiplicative (%)

| Schedule | AW dev | AW test | AW challenge | EO test |
|---|---|---|---|---|
| *Pass@4* | | | | |
| Annealed $1 \to 0$ | $\mathbf{67.98} \pm 2.21$ | $\mathbf{78.72} \pm 1.71$ | $49.10 \pm 0.66$ | $\mathbf{37.50} \pm 1.42$ |
| Fixed $\lambda = 1$ | $\mathbf{67.98} \pm 0.88$ | $77.83 \pm 1.97$ | $\mathbf{50.84} \pm 0.81$ | $35.94 \pm 1.10$ |
| *Pass@1* | | | | |
| Annealed $1 \to 0$ | $45.29 \pm 1.80$ | $\mathbf{54.13} \pm 1.95$ | $26.82 \pm 0.30$ | $\mathbf{24.17} \pm 1.09$ |
| Fixed $\lambda = 1$ | $\mathbf{47.57} \pm 1.84$ | $54.04 \pm 0.67$ | $\mathbf{28.43} \pm 0.73$ | $23.29 \pm 0.63$ |

Annealing reduces $\lambda$ linearly from one to zero; fixed weighting holds it at one. Mean $\pm$ SD over four seeds.

On EnterpriseOps, annealing reaches 37.50% pass@4 and 24.17% pass@1, compared with 35.94% and 23.29% for fixed weighting. The differences are 1.56 and 0.88 percentage points. Both schedules improve pass@4 over GRPO's 33.01%, so procedural weighting remains useful even without annealing.

On AppWorld, both schedules reach 67.98% development pass@4. Fixed weighting has the higher development pass@1, 47.57% versus 45.29%. On test-168, annealing is slightly higher: 78.72% versus 77.83% pass@4, and 54.13% versus 54.04% pass@1. Both preserve a large improvement over GRPO's 59.67% pass@4. On challenge-417, fixed weighting leads by 1.74 percentage points on pass@4 (50.84% versus 49.10%) and 1.61 percentage points on pass@1 (28.43% versus 26.82%).

The main results consistently report the annealed multiplicative policy. The fixed-schedule comparison shows that the large AppWorld gains do not require the procedural signal to fade, while EnterpriseOps benefits from annealing in these runs. Neither schedule leads on every split and metric.

## E Procedural Connections to External Tasks

The transfer results raise a concrete question: what do tasks in different environments have in common? We examine behaviors that recur across the training procedures and external benchmarks,

such as resolving a reference before using it, checking a precondition before changing a record, and completing every part of a request. Table 14 collects the relevant score comparisons. Appendix A provides the original procedure texts and their selection counts.

The task examples below are paraphrased from public BFCL and $\tau^3$ definitions (Gorilla Team, 2026; Sierra Research, 2026a). They illustrate the demands of the categories, rather than presenting successful rollouts from our checkpoints. The correspondence between a procedure and a gain is a qualitative interpretation: external token-level scoring differences and individual-procedure removal experiments were not recorded. All reported external evaluations use the trained policy without hints.

Table 14: Selected external comparisons (%)

| Category / domain | GRPO | PHS-Multiplicative | PHS-Additive | Δ Mult. | Δ Add. |
|---|---|---|---|---|---|
| *Training on EnterpriseOps* | | | | | |
| BFCL: parallel (non-live) | 73.00 | 83.00 | **83.88** | +10.00 | +10.88 |
| BFCL: parallel multiple (non-live) | 79.25 | **85.00** | 84.50 | +5.75 | +5.25 |
| BFCL: long context (multi-turn) | 61.12 | **69.50** | 65.25 | +8.38 | +4.13 |
| BFCL: missing function (multi-turn) | 57.38 | **65.12** | 63.38 | +7.74 | +6.00 |
| $\tau^3$: airline | 74.50 | **84.00** | 80.31 | +9.50 | +5.81 |
| $\tau^3$: retail | 82.17 | 84.02 | **86.96** | +1.85 | +4.79 |
| $\tau^3$: telecom | 67.32 | 71.05 | **73.69** | +3.73 | +6.37 |
| *Training on AppWorld* | | | | | |
| BFCL: relevance detection (live) | 76.56 | 87.50 | **89.06** | +10.94 | +12.50 |
| BFCL: parallel (live) | 64.06 | **75.00** | 71.88 | +10.94 | +7.82 |
| BFCL: simple (non-live) | 70.35 | **74.34** | 74.08 | +3.99 | +3.73 |
| $\tau^3$: retail | 81.28 | 86.56 | **88.38** | +5.28 | +7.10 |
| $\tau^3$: airline | 76.00 | 79.29 | **83.50** | +3.29 | +7.50 |

BFCL: category-level single-attempt accuracy; $\tau^3$: pass@1. Means over four seeds; Δ is the gain over same-source GRPO in points. Selected comparisons, not an aggregate; full scores and SDs are in Appendix F.

### E.1 Completing every requested call

EnterpriseOps-trained policies improve BFCL non-live parallel accuracy from 73.00% under GRPO to 83.00% with PHS-Multiplicative and 83.88% with PHS-Additive. Parallel-multiple accuracy rises from 79.25% to 85.00% and 84.50%.

In BFCL's `parallel_0`, the user requests Spotify playback for two artists, with a different duration for each. The supplied function accepts one artist and one duration, so satisfying the request requires two calls with the corresponding argument pairs. Parallel categories check the set of calls and their arguments, not just whether the model emits any call (Patil et al., 2025; Gorilla Team, 2026).

The EnterpriseOps procedure *Execute every required mutation*, selected for 247 of 441 training tasks, decomposes a request into all required changes and checks that each is performed. The shared requirement is coverage: completing one part does not complete the request. The procedure originated in record updates; the benchmark example expresses the same requirement through independent playback calls. AppWorld-trained policies do not show the same non-live parallel gain: their scores are 80.38% and 80.12%, compared with 81.62% for GRPO. Thus this category's improvement is specific to the training environment and method combination.

### E.2 Maintaining references and recognizing missing tools

On BFCL multi-turn long context, EnterpriseOps-trained PHS-Multiplicative improves over GRPO by 8.38 percentage points, from 61.12% to 69.50%. On the missing-function category, the gain is 7.74 percentage points, from 57.38% to 65.12%. The additive gains are 4.13 and 6.00 percentage points.

The public `multi_turn_long_context_0` task first moves a report into a newly created directory. Later turns ask the agent to search it, sort its contents, and compare it with an earlier report. The agent must carry the updated file location into the later calls despite a large initial file inventory. BFCL checks both the environment state and the required response path after each turn (Gorilla

Team, 2024). The related missing-function example initially withholds the sorting tool and supplies it later. The agent must recognize the gap before it can complete that part of the workflow.

*Resolve references before writing* (114 training tasks) and *Thread created ids into follow-ups* (118) both teach the broader dependency pattern of obtaining a concrete reference and using it in subsequent actions. The file example uses paths rather than database identifiers. *Confirm gating condition before acting* (112) provides a related check-before-action pattern for a step whose required tool is unavailable. These are shared procedural demands across different interfaces; missing-function success also requires responding correctly when the tool becomes available.

### E.3 Checking whether an airline action is permitted

EnterpriseOps-trained PHS-Multiplicative raises airline pass@1 from 74.50% to 84.00%, a gain of 9.50 percentage points. The PHS-Additive policy reaches 80.31%. Multiplicative pass$^4$ also rises, from 62.00% to 72.00% (Table 3).

Airline task `0` asks to cancel a booking outside the usual cancellation window. The user argues that insurance purchased for an earlier trip should suffice and does not want cancellation without a refund. The task definition expects the agent to refuse the disallowed cancellation. The airline policy requires checking the reservation's eligibility before calling the cancellation API; the API itself does not enforce these conditions (Sierra Research, 2026a).

This task makes the relevance of *Confirm gating condition before acting* explicit. The procedure asks the model to inspect the governing record, evaluate the condition, and choose the appropriate branch. Knowing how to call the cancellation tool is only one part of solving the task; knowing when the call is permitted is another. The AppWorld policies also improve airline pass@1, by 3.29 and 7.50 percentage points, so the gain is not exclusive to a catalog with this particular entry.

### E.4 Choosing the right items and finishing a state change

EnterpriseOps-trained additive policies improve retail pass@1 by 4.79 percentage points, from 82.17% to 86.96%, and telecom by 6.37 percentage points, from 67.32% to 73.69%. Multiplicative policies improve these domains by 1.85 and 3.73 percentage points.

Retail task `0` requests two exchanges: a keyboard with specified switch, size, and lighting attributes, and a thermostat compatible with a different home platform. It also specifies a fallback if the preferred keyboard is unavailable. Completing the task requires checking candidate attributes and including both items in the exchange. The reference action uses one batched exchange call containing both items (Sierra Research, 2026a). The transferable requirement is complete coverage, which can be implemented differently by different APIs. *Select candidate matching task subject* (82 tasks) addresses the attribute comparison, while *Execute every required mutation* (247) addresses completing the full request.

A telecom example involves a customer abroad with roaming disabled both on the service line and on the phone. The reference solution enables roaming on the account and has the user enable it on the device; the task's environment assertions then check restored mobile data and connection quality (Sierra Research, 2026a). One successful change is insufficient. *Continue through entire task chain* (75 tasks) and *Confirm phase transition persisted* (67) emphasize completing the dependent actions and checking the resulting state. The AppWorld entry *Verify the result and finalize with complete_task* (38 of 82 tasks) expresses a related verification principle through its own interface.

### E.5 Using available functions on unfamiliar interfaces

AppWorld-trained policies improve live BFCL relevance detection from 76.56% for GRPO to 87.50% and 89.06%. Live parallel accuracy improves from 64.06% to 75.00% and 71.88%. GRPO is below the untrained base in both categories, whose base scores are 81.25% and 71.88%, respectively.

The live parallel example `live_parallel_0-0-0` asks for current weather in Beijing and Shanghai. A single-location weather function is available, requiring one call per city. The live relevance example `live_relevance_0-0-0` requests an illustration and supplies several tools,

including image generation. Here the relevance evaluator checks for a nonempty, decodable function call; it does not establish that the chosen function or arguments are correct (Gorilla Team, 2026).

The AppWorld entry *Match task vocabulary to the exact dedicated endpoint instead of a plausible proxy* (11 training tasks) encourages inspecting available functions and relating them to the requested operation. This is a plausible connection to using an available tool, although relevance accuracy alone cannot measure exact endpoint matching. Likewise, *Exhaustively paginate every list endpoint* (55) emphasizes covering a complete set, but pagination is not itself required by the two-city task. These examples show how an abstract procedural requirement can recur without the original API operation recurring.

### E.6 From correspondence to a measured mechanism

The pattern is not a one-to-one map from catalog titles to benchmark gains. AppWorld-trained PHS-Additive telecom pass@1 improves by 1.53 percentage points, while the multiplicative policy declines by 0.22. Both AppWorld policies improve long-context BFCL accuracy, by 3.75 and 4.00 percentage points. EnterpriseOps-trained relevance detection declines by 3.13 percentage points with PHS-Multiplicative but improves by 4.68 with PHS-Additive. Procedure content, objective choice, and training environment can all matter.

The additive controls in Section 5.5 test whether task-specific guidance matters in aggregate. Attributing a particular external gain to a particular procedure would require a finer intervention: remove that entry from the training map, retrain with a matched budget, and evaluate the corresponding categories. Re-scoring saved external trajectories with and without that procedure could also locate the tokens it favors. Together, such experiments would test whether the behavioral connections illustrated here explain the observed transfer.

## F Complete Reported Results

As in Section 5, blue and orange means are above and below GRPO (the no-hint prompt in selector tables), grey values are uncertainties, and bold marks the highest mean. This section reports the detailed in-domain, cross-environment, external, and control evaluations. Each table identifies its metric, units, and uncertainty convention.

### F.1 In-domain and cross-environment evaluations

Table 15: Held-out success, pass@1

| Arm | AW dev | AW test | AW challenge | EO test |
|---|---|---|---|---|
| Base | 15.04 ± 0.69 | 21.32 ± 0.67 | 6.90 ± 0.19 | 15.89 ± 0.76 |
| GRPO | 28.27 ± 3.05 | 36.55 ± 0.64 | 16.24 ± 0.22 | 23.19 ± 1.21 |
| OPSD | 20.20 ± 1.05 | 35.04 ± 0.44 | 12.58 ± 0.65 | 1.66 ± 0.25 |
| PHS-Multiplicative | **45.29** ± 1.80 | **54.13** ± 1.95 | **26.82** ± 0.30 | 24.17 ± 1.09 |
| PHS-Additive | 41.56 ± 2.10 | 50.07 ± 0.64 | 24.01 ± 0.43 | **24.90** ± 1.20 |

Mean ± SD over four seeds. Pass@4 is in Table 1.

Table 16: Cross-environment transfer, pass@1

| Arm | EO test | AW dev | AW test | AW challenge |
|---|---|---|---|---|
| Base | 15.89 ± 0.76 | 15.04 ± 0.69 | 21.32 ± 0.67 | 6.90 ± 0.19 |
| GRPO | 15.28 ± 0.46 | 11.01 ± 1.09 | 21.27 ± 2.07 | 6.93 ± 0.30 |
| OPSD | 18.80 ± 1.36 | 12.39 ± 0.70 | 20.93 ± 0.59 | 7.51 ± 0.20 |
| PHS-Multiplicative | **18.85** ± 0.67 | **17.31** ± 1.09 | 25.61 ± 1.32 | **9.45** ± 0.38 |
| PHS-Additive | 17.72 ± 0.89 | 17.30 ± 1.04 | **26.45** ± 0.51 | 9.41 ± 0.23 |

First column: AppWorld-trained; others: EnterpriseOps-trained. Mean ± SD over four seeds. Pass@4 is in Table 2.

### F.2 External evaluation over four attempts

BFCL pass@4 and $\tau^3$ pass$^4$ are in Tables 3 and 4; $\tau^3$ pass@4 follows.

Table 17: $\tau^3$ pass@4 (%)

| | AppWorld-trained | | | EnterpriseOps-trained | | |
|---|---|---|---|---|---|---|
| Arm | Retail | Airline | Telecom | Retail | Airline | Telecom |
| Base | 96.49 ± 1.72 | 84.00 ± 5.18 | 78.95 ± 3.82 | 96.49 ± 1.72 | 84.00 ± 5.18 | 78.95 ± 3.82 |
| GRPO | 97.37 ± 1.50 | 86.00 ± 4.91 | **81.58** ± 3.63 | 96.49 ± 1.72 | 84.00 ± 5.18 | 77.19 ± 3.93 |
| OPSD | 97.37 ± 1.50 | 88.00 ± 4.60 | 77.19 ± 3.93 | 84.21 ± 3.42 | 58.00 ± 6.98 | 73.68 ± 4.12 |
| PHS-Multiplicative | 97.37 ± 1.50 | **92.00** ± 3.84 | 80.70 ± 3.70 | 95.61 ± 1.92 | **92.00** ± 3.84 | **79.82** ± 3.76 |
| PHS-Additive | **100.00** ± 0.00 | **92.00** ± 3.84 | **81.58** ± 3.63 | **97.37** ± 1.50 | 90.00 ± 4.24 | **79.82** ± 3.76 |

Tasks solved in at least one of four attempts, ± task-level standard error. Retail, airline and telecom have 114, 50 and 114 tasks.

### F.3 Single-attempt external evaluations

The EnterpriseOps-trained OPSD checkpoint emits empty or code-free turns in 14–48% of its AppWorld and $\tau^3$ attempts, compared with at most 6% for the other arms. These OPSD attempts are counted as failures. This is a behavior observation, not evidence that the cause matches a particular distillation failure mechanism.

Table 18: $\tau^3$ single-attempt success (%)

| | AppWorld-trained | | | EnterpriseOps-trained | | |
|---|---|---|---|---|---|---|
| Arm | Retail | Airline | Telecom | Retail | Airline | Telecom |
| Base | 82.86 ± 0.86 | 70.35 ± 3.46 | 69.08 ± 0.44 | 82.86 ± 0.86 | 70.35 ± 3.46 | 69.08 ± 0.44 |
| GRPO | 81.28 ± 1.60 | 76.00 ± 1.63 | 72.15 ± 2.07 | 82.17 ± 2.84 | 74.50 ± 1.00 | 67.32 ± 1.81 |
| OPSD | 83.11 ± 0.44 | 78.40 ± 1.82 | 67.11 ± 1.13 | 67.11 ± 4.21 | 37.50 ± 1.91 | 57.68 ± 1.10 |
| PHS-Multiplicative | 86.56 ± 1.98 | 79.29 ± 1.89 | 71.93 ± 1.43 | 84.02 ± 2.02 | **84.00** ± 4.00 | 71.05 ± 1.01 |
| PHS-Additive | **88.38** ± 2.62 | **83.50** ± 1.91 | **73.68** ± 0.00 | **86.96** ± 2.71 | 80.31 ± 0.90 | **73.69** ± 2.03 |

Mean ± SD over four seeds.

Table 19: AppWorld checkpoints: category-level single-attempt BFCL accuracy (%)

| BFCL bucket | Base | GRPO | PHS-Multiplicative | PHS-Additive |
|---|---|---|---|---|
| *Non-live* | | | | |
| Simple | 71.08 ± 0.24 | 70.35 ± 0.37 | **74.34** ± 0.88 | 74.08 ± 0.48 |
| Multiple | 92.12 ± 0.25 | 92.50 ± 0.41 | 93.12 ± 0.48 | **93.50** ± 0.71 |
| Parallel | **81.62** ± 0.48 | **81.62** ± 0.48 | 80.38 ± 1.03 | 80.12 ± 0.25 |
| Parallel multiple | 81.38 ± 0.63 | 81.25 ± 0.29 | **83.88** ± 0.48 | 82.62 ± 0.63 |
| *Multi-turn* | | | | |
| Base | 68.88 ± 1.60 | 68.88 ± 1.70 | 69.75 ± 1.50 | **70.38** ± 1.11 |
| Missing function | 55.75 ± 1.55 | 58.00 ± 1.15 | 60.00 ± 1.78 | **60.12** ± 0.85 |
| Missing parameter | 49.38 ± 1.31 | 48.38 ± 0.63 | **50.88** ± 2.32 | 49.25 ± 1.44 |
| Long context | 61.75 ± 0.96 | 60.00 ± 1.96 | 63.75 ± 1.19 | **64.00** ± 0.71 |
| *Live* | | | | |
| Simple | 81.40 ± 0.71 | 80.43 ± 0.23 | **82.75** ± 0.92 | 80.81 ± 0.67 |
| Multiple | 79.44 ± 0.28 | 79.63 ± 0.25 | **80.36** ± 0.40 | 80.34 ± 0.13 |
| Parallel | 71.88 ± 3.61 | 64.06 ± 3.12 | **75.00** ± 0.00 | 71.88 ± 3.61 |
| Parallel multiple | 73.96 ± 2.09 | 73.96 ± 2.09 | 79.17 ± 0.00 | **80.21** ± 2.08 |
| Relevance detection | 81.25 ± 5.10 | 76.56 ± 3.12 | 87.50 ± 5.10 | **89.06** ± 3.12 |

Mean ± SD across four seeds. These are distinct from aggregate pass@4; OPSD category summaries are unavailable.

Table 20: EnterpriseOps checkpoints: category-level single-attempt BFCL accuracy (%)

| BFCL bucket | Base | GRPO | PHS-Multiplicative | PHS-Additive |
|---|---|---|---|---|
| *Non-live* | | | | |
| Simple | 71.08 ± 0.24 | 72.02 ± 0.69 | **74.27** ± 0.71 | 72.81 ± 0.23 |
| Multiple | 92.12 ± 0.25 | 94.12 ± 0.25 | 94.50 ± 0.41 | **94.62** ± 0.85 |
| Parallel | 81.62 ± 0.48 | 73.00 ± 1.78 | 83.00 ± 1.00 | **83.88** ± 0.25 |
| Parallel multiple | 81.38 ± 0.63 | 79.25 ± 0.65 | **85.00** ± 0.41 | 84.50 ± 0.41 |
| *Multi-turn* | | | | |
| Base | 68.88 ± 1.60 | 69.62 ± 1.11 | **73.88** ± 1.25 | 73.25 ± 2.40 |
| Missing function | 55.75 ± 1.55 | 57.38 ± 0.95 | **65.12** ± 1.65 | 63.38 ± 0.48 |
| Missing parameter | 49.38 ± 1.31 | 48.38 ± 0.25 | 52.00 ± 1.41 | **52.25** ± 0.29 |
| Long context | 61.75 ± 0.96 | 61.12 ± 1.89 | **69.50** ± 1.08 | 65.25 ± 1.04 |
| *Live* | | | | |
| Simple | 81.40 ± 0.71 | 80.62 ± 0.00 | **83.53** ± 1.02 | 83.24 ± 0.80 |
| Multiple | 79.44 ± 0.28 | 79.03 ± 0.38 | **79.94** ± 0.14 | 79.86 ± 0.11 |
| Parallel | 71.88 ± 3.61 | 67.19 ± 3.12 | **73.44** ± 3.12 | 68.75 ± 0.00 |
| Parallel multiple | 73.96 ± 2.09 | 72.91 ± 2.41 | **77.09** ± 2.41 | 75.00 ± 0.00 |
| Relevance detection | 81.25 ± 5.10 | 84.38 ± 3.61 | 81.25 ± 0.00 | **89.06** ± 3.12 |

Mean ± SD across four seeds. These are distinct from aggregate pass@4; OPSD category summaries are unavailable.

### F.4 Additive training controls

Table 21: EnterpriseOps additive training controls (%)

| Condition | Pass@4 | Pass@1 |
|---|---|---|
| GRPO | 33.01 ± 0.39 | 23.19 ± 1.21 |
| Generic procedure | 29.88 ± 0.75 | 20.51 ± 0.66 |
| Reassigned map | 33.01 ± 2.33 | 21.73 ± 0.79 |
| Aligned map | **40.43** ± 1.73 | **24.90** ± 1.20 |

Pass@1 and pass@4 are means ± SD across four seeds.

Task-matched procedures exceed GRPO by 7.42 percentage points on pass@4 and 1.71 percentage points on pass@1. Reassignment matches GRPO on pass@4 and falls below it by 1.46 percentage points on pass@1. Generic guidance falls below GRPO by 3.13 and 2.68 percentage points, respectively. Task-matched procedures also exceed reassigned ones by 3.17 percentage points on pass@1.

### F.5 Evaluation-only selector diagnostics

The selector diagnostic measures immediate prompting effects on the frozen base, separately from policy training. It compares no hint, random catalog entries, the selector without training-task examples, the selector with them, and the learner itself selecting with the same prompt.

Pass@4 is in Table 6; the ordering there does not hold for every pass@1 comparison below. The learner-as-selector condition evaluates selection at prompting time, not training with a learner-selected map. Training-task diagnostics show a similar benefit from the examples (Table 23). The EnterpriseOps no-hint baseline differs from the main census; gains are compared within each campaign.

Table 22: Prompting the untrained model with hints, pass@1

| Hints | AW dev | AW test | AW challenge | EO test |
|---|---|---|---|---|
| No hint | 15.04 ± 0.69 | 21.32 ± 0.67 | 6.90 ± 0.19 | 22.75 ± 0.25 |
| Random procedures | 16.29 ± 0.53 | 24.83 ± 1.77 | 9.05 ± 0.15 | 23.97 ± 0.54 |
| Selector, no examples | 17.12 ± 1.08 | 29.33 ± 0.18 | 9.52 ± 0.26 | 23.88 ± 0.89 |
| Selector with training examples (PHS) | 18.48 ± 0.81 | 29.15 ± 1.37 | 10.36 ± 0.32 | 25.44 ± 0.58 |
| Learner as selector | **20.86** ± 2.26 | **30.50** ± 0.98 | **10.99** ± 0.56 | **28.17** ± 0.29 |

No policy training; mean ± SD over four seeds. Pass@4 is in Table 6.

The same diagnostic on training tasks evaluates the hint maps in the population for which they were selected, without optimizing the learner. The selector with examples improves pass@4 from 43.29% to 47.26% on AppWorld and from 44.16% to 45.58% on EnterpriseOps compared with the selector without them. The learner-selected maps reach 48.17% and 44.67%. These observations do not establish the training performance of a policy optimized with those alternative maps.

Table 23: Prompting the untrained model on training tasks (%)

| | AppWorld train-82 | | EnterpriseOps train-441 | |
|---|---|---|---|---|
| Hints | Pass@4 | Pass@1 | Pass@4 | Pass@1 |
| No hint | 35.98 ± 1.22 | 15.94 ± 0.45 | 42.18 ± 0.89 | 27.41 ± 0.79 |
| Random procedures | 39.33 ± 1.53 | 18.50 ± 1.31 | 42.74 ± 1.55 | 28.88 ± 1.08 |
| Selector, no examples | 43.29 ± 1.57 | 22.72 ± 0.77 | 44.16 ± 1.02 | 30.56 ± 0.88 |
| Selector with training examples (PHS) | 47.26 ± 3.50 | 23.94 ± 1.67 | **45.58** ± 0.56 | **31.18** ± 0.43 |
| Learner as selector | **48.17** ± 1.22 | **26.17** ± 0.93 | 44.67 ± 0.37 | **31.18** ± 0.53 |

The selector-with-examples row uses the training hint map. No policy training; mean ± SD over four seeds.