LATENT ON-POLICY SELF-DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Enabling agents to learn from experience and internalize it into their policy has become a central problem in self-evolving AI. On-policy self-distillation (OPSD) offers an effective pathway by using a privileged self-teacher to provide dense supervision on the student's own trajectories; however, existing methods still rely heavily on designer-specified privileged artifacts (*e.g.*, answers, feedback, skills, or trajectories), limiting the end-to-end learnability and scalability required for continual self-improvement. In this work, we introduce **Latent On-Policy Self-Distillation** (LOPD), which, rather than proposing another hand-crafted OPSD variant with a newly prescribed form of privileged context, makes the teacher's privileged context itself learnable end-to-end from experience. Technically, LOPD retrieves relevant experiences and composes them into continuous latent tokens that condition a self-teacher, while the student generates trajectories from the task and interaction history and receives dense token-level supervision at every visited prefix. We further introduce a privileged-margin objective to stabilize and regulate the learning of latent context. Empirically, LOPD demonstrates **(I) strong performance**, outperforming RLVR and representative OPSD methods including OPSD, SDPO, and Skill-SD across both agentic tool use and code generation; and **(II) high learning efficiency**, surpassing GRPO and Skill-SD with less than 30% of their rollout budget. Ablation studies further provide direct evidence that making privileged context learnable is necessary for realizing these gains. Together, these results position LOPD as a step toward a more scalable and self-directed paradigm for agent evolution. Code is anonymously available here.

1 INTRODUCTION

How should an agent learn from experience? A natural answer is to expose the model to prior successful trajectories, corrective feedback, or expert demonstrations, and then train it to internalize the behaviors they reveal. This intuition has made on-policy distillation (OPD) a central paradigm for experience learning in large language models: the student first samples its own trajectory, and a teacher then provides dense token-level supervision on the states the student actually visits (Song & Zheng, 2026; Li et al., 2026b). Compared with off-policy imitation, OPD reduces the mismatch between training and inference; compared with reinforcement learning with verifiable rewards (RLVR), it converts sparse outcome feedback into fine-grained distributional signals over the student's own rollout (Hübotter et al., 2026; Yang et al., 2026). In this view, OPD is not merely a compression technique for a stronger model, but a mechanism for turning external experience into persistent policy improvement.

On-policy self-distillation (OPSD) has recently emerged as an especially appealing form of this paradigm because it removes the dependence on a separate, stronger teacher. Instead, the teacher and the student are instantiated from the same model under different contexts: the student acts from the task and interaction history, while the teacher additionally receives privileged information such as verified reasoning traces, final answers, environment feedback, or successful prior rollouts (Zhao et al., 2026; Hübotter et al., 2026; Penaloza et al., 2026). The privileged context makes the teacher distribution more informative than the student's, enabling dense feedback without querying an external model. This design shifts the central question of OPSD from *which teacher is strong enough?* to *what privileged context should be given to the self-teacher?* The answer matters: the context must expose useful experience, make token-level supervision sharper, and yet remain compatible with a student that will not see that context at inference time.

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.

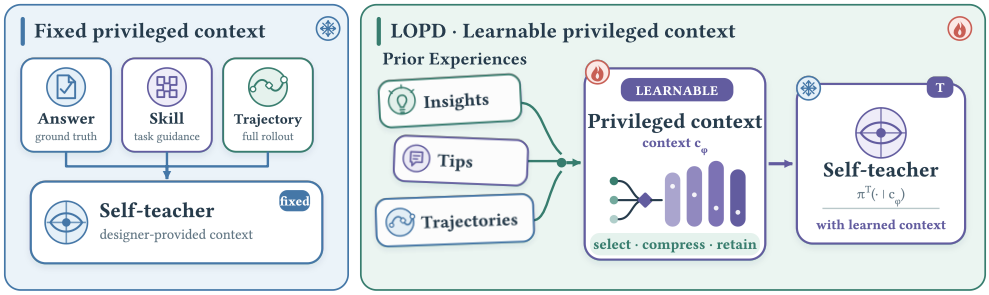

Figure 1: **From fixed to learnable privileged context.** Existing OPSD methods condition the self-teacher on designer-specified answers, skills, or trajectories. LOPD instead learns latent context from prior experience and distills its dense supervision into the same on-policy student. Insights/tips illustrate compatible sources; LOPD instantiates the substrate using trajectories only.

Existing substrates for privileged experience remain imperfect in a more fundamental sense. Although prior work instantiates privileged context in many forms—verified answers, reasoning traces, environment feedback, contrastive evidence, or action-only trajectories (Zhao et al., 2026; Hübotter et al., 2026; Yu et al., 2026c; Penaloza et al., 2026; Yang et al., 2026)—these contexts are typically hand-designed or rule-extracted transformations of experience. They decide *a priori* what the teacher should see: an oracle answer, a textual reflection, a successful rollout, a retrieved document, or another discrete artifact. Such substrates can be useful, but they also constrain self-distillation to the information format chosen by the designer, rather than allowing the teacher to learn which aspects of prior experience are actually useful for supervising the student's current trajectory. This motivates a substrate-level question for OPSD:

Research Question

Can privileged context itself be learned end-to-end from experience, so that the self-teacher, rather than a designer-specified rule, determines what experiential knowledge to retain and how to encode it for dense on-policy supervision?

To address this challenge, we propose **Latent On-Policy Self-Distillation** (LOPD), a framework that realizes privileged context as a *learnable latent substrate*. Its training pipeline otherwise follows standard OPSD: the student first rolls out trajectories from the task and interaction history, and the teacher re-evaluates every visited prefix to provide dense token-level distributions, against which the student is optimized through reverse-KL distillation. The key difference is that the teacher is conditioned not on a pre-defined privileged artifact, but on learnable latent context instantiated by a composer that transforms retrieved experiences into compact continuous tokens. Because this context is differentiable, the same training process that improves the student also teaches the composer which aspects of experience to retain and how to organize them into effective teacher supervision. To ensure that this learning makes the teacher more informative rather than merely easier for the student to match, we introduce a privileged-margin constraint that requires the teacher to maintain a verifiable log-probability advantage over the student. After training, only the student is retained: inference requires no experience database, retrieval module, composer, or latent context. In this way, LOPD turns experience from a hand-crafted input artifact into an end-to-end learnable supervision substrate whose benefits are internalized by the student policy.

Positioning

This work does not aim to introduce yet another elaborate OPSD variant by hand-designing a new form of privileged context. Its central claim is at the level of the *experience substrate*: if self-evolving AI is to scale, neither the useful content of experience nor its representation should be fixed by designer heuristics. A scalable system should learn end-to-end what evidence to preserve and how to transform it into supervision. We study OPSD as a concrete mechanism for realizing this broader principle.

Our contributions are summarized as follows:

- **Context Re-formulation.** We reformulate privileged context from pre-defined, hand-designed textual artifacts into an end-to-end learnable latent substrate, allowing the teacher to automatically extract task-relevant supervision signals from prior experience.

- **Proposed Solution.** We develop LOPD, which composes retrieved experiences into continuous latent context for the self-teacher and jointly optimizes the privileged context and student through on-policy distillation. A privileged-margin constraint keeps the learned context informative and prevents the teacher from collapsing toward the student.
- **Experimental Validation.** LOPD consistently surpasses RLVR and representative OPSD methods over three model backbones and seven benchmarks, and outperforms GRPO and Skill-SD with less than 30% of their rollout budget. Further ablation studies directly establish the necessity of jointly learning the privileged context.

2 Related Work

On-Policy (Self-)Distillation. On-policy distillation trains a student on trajectories sampled from its own policy while querying a teacher for dense token-level supervision on the visited states (Song & Zheng, 2026; Li et al., 2026b). Recent work has applied this recipe broadly: reasoning and efficient post-training (Wu et al., 2026a; Jin et al., 2026; Fu et al., 2026), long-context modeling (Zhang et al., 2026a), context and knowledge distillation (Ye et al., 2026; Lazaridis et al., 2026), GUI grounding (Zhang et al., 2026b), and multimodal domains such as video grounding (Li et al., 2026a), speech alignment (Cao et al., 2026), and visual reasoning (Yuan et al., 2026; Liu et al., 2026). Within this landscape, on-policy self-distillation (OPSD) removes the external teacher by instantiating teacher and student from the same model under different contexts: the student acts under the deployable input, while the teacher receives privileged context (Zhao et al., 2026; Hübotter et al., 2026; Penaloza et al., 2026; Yang et al., 2026; Yu et al., 2026c). This line is best understood by the form of privileged context it supplies: verified answers and reasoning traces (Zhao et al., 2026; Yang et al., 2026), rich environment feedback or self-revision signals (Hübotter et al., 2026; He et al., 2026; Zhang et al., 2026c), contrastive or peer-rollout evidence (Yu et al., 2026a; Pan et al., 2026), action-only frontier-agent trajectories (Penaloza et al., 2026), and retrieved or evidence-guided context (Ye et al., 2026; Lazaridis et al., 2026). These designs demonstrate that privileged context is the key interface through which OPSD converts experience into supervision, but the context itself is usually pre-defined as a textual or discrete artifact; in contrast, LOPD makes the privileged context a learnable latent substrate jointly optimized with distillation.

Latent Computation. Latent computation uses continuous latent tokens/embeddings or hidden states as the carrier of LLM computation rather than natural language (Yu et al., 2026b; Zhu et al., 2025). In *reasoning*, latent tokens can expand the model's internal compute budget (Deng et al., 2026; Hao et al., 2025; Amos et al., 2026), letting the model perform deeper deliberation before producing an answer. In *memory*, latent states can serve as compact carriers of procedural (Zhang et al., 2025; Yu et al., 2026d), factual (Wang et al., 2024; Feng et al., 2026), and experiential (Hou et al., 2026; Wu et al., 2026b) information, preserving reusable experience without exposing long textual traces in the prompt. In *planning*, latent computation can represent intermediate plans, subgoals, or world-state summaries that guide downstream actions while remaining flexible and differentiable (Zheng & Lee, 2026; Hanna & Ameisen, 2026). Our method is conceptually close to latent memory, but uses it in a different role: the latent state is not an inference-time augmentation, but a learnable privileged context through which an on-policy teacher supervises a student that does not observe this context.

3 Method

3.1 Problem Setup: Privileged Context in OPSD

We consider a multi-turn agent that receives a task x and interacts with an environment through observations and actions. We call the acting policy the *student*. At turn t, the student observes $\boldsymbol{s}_t = (x, \boldsymbol{o}_{\le t}, \boldsymbol{a}_{<t})$, where $\boldsymbol{o}_{\le t}$ denotes observations so far and $\boldsymbol{a}_{<t}$ denotes previous actions, and samples actions to form a trajectory:

$$\boldsymbol{a}_t \sim \pi_\theta^S(\cdot \mid \boldsymbol{s}_t), \qquad \boldsymbol{\tau} = (\boldsymbol{s}_1, \boldsymbol{a}_1, \ldots, \boldsymbol{s}_{|\boldsymbol{\tau}|}, \boldsymbol{a}_{|\boldsymbol{\tau}|}) \sim \pi_\theta^S(\cdot \mid x), \tag{1}$$

where $\boldsymbol{a}_t = (a_{t,1}, \ldots, a_{t,L_t})$ is the tokenized action at turn t, L_t is its length, and $|\boldsymbol{\tau}|$ is the number of turns. OPSD constructs a teacher by giving the same model additional privileged context. Abstractly, this context is obtained from some experience source $\mathcal{E}$ by a fixed transformation:

$$\boldsymbol{c}_{\text{fix}} = \Phi_{\text{fix}}(x, \mathcal{E}), \qquad \pi_\theta^T(\cdot \mid \boldsymbol{s}_t, \boldsymbol{c}_{\text{fix}}) = \pi_\theta(\cdot \mid [\boldsymbol{s}_t; \boldsymbol{c}_{\text{fix}}]), \tag{2}$$

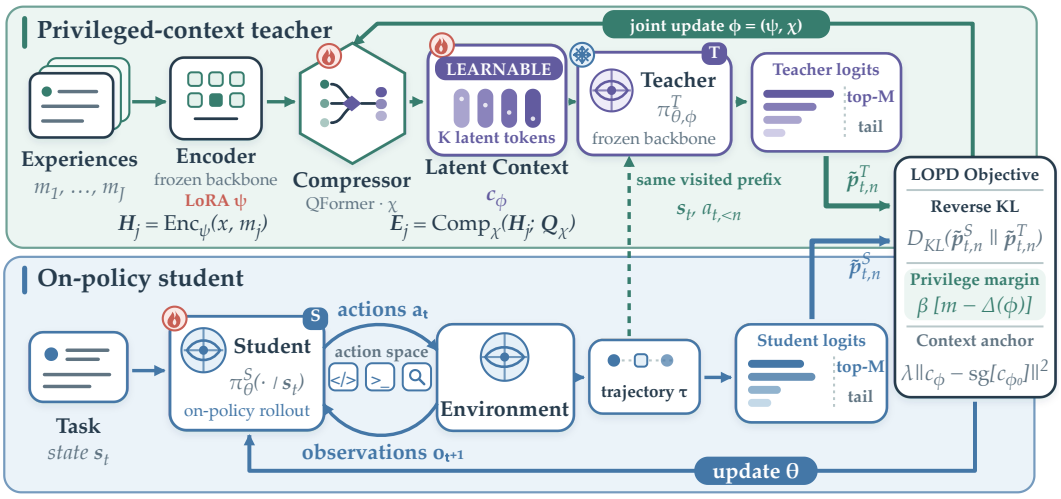

Figure 2: **Overview of LOPD.** The student generates on-policy trajectories from the task and interaction history. A fixed-backbone teacher re-evaluates the same prefixes with latent context composed from retrieved experiences, and reverse-KL distillation matches their top-M-plus-tail distributions.

where $\mathcal{E}$ may contain answers, traces, feedback, demonstrations, or retrieved trajectories, and Φ_{fix} is a designer-specified rule that decides what artifact the teacher sees. Given a student trajectory, OPSD matches teacher and student distributions on the same visited prefixes:

$$\mathcal{L}_{\text{OPSD}}(\theta) = \mathbb{E}_{\boldsymbol{\tau}\sim\pi^S_\theta}\left[\sum_{t=1}^{|\boldsymbol{\tau}|}\sum_{n=1}^{L_t} D\big(\mathrm{sg}\big[\pi^T_\theta(\cdot \mid \boldsymbol{s}_t, \boldsymbol{c}_{\text{fix}}, a_{t,<n})\big] \,\big\|\, \pi^S_\theta(\cdot \mid \boldsymbol{s}_t, a_{t,<n})\big)\right], \tag{3}$$

where $a_{t,<n}$ is the action prefix before token n, D is a token-level divergence, and $\mathrm{sg}[\cdot]$ denotes stop-gradient. This setup makes the limitation explicit: the supervision quality is bounded by a pre-defined context constructor. We instead parameterize the constructor:

$$\boldsymbol{c}_\phi = \Phi_\phi(x, \mathcal{E}) = \langle e_1\rangle \oplus \langle e_2\rangle \oplus \cdots \oplus \langle e_K\rangle, \tag{4}$$

where Φ_ϕ is a learnable composer, $\langle e_i\rangle$ is a continuous latent token, K is the number of latent tokens per retrieved experience, and $\oplus$ denotes context concatenation. The goal of LOPD is to make privileged context itself learnable while preserving the defining asymmetry of OPSD: the teacher additionally observes $\boldsymbol{c}_\phi$, whereas the student continues to condition only on $\boldsymbol{s}_t$.

3.2 Learnable Latent Privileged Context

LOPD instantiates Φ_ϕ as a latent-context composer. We deliberately begin with a minimal experience pipeline. Offline, we retain successful rollouts in an experience bank, with each entry storing its task description and a compact action–result trace that omits verbose observations. For a current task x, a dense retriever embeds the query and the task description of each stored entry, then returns the top-J entries under cosine similarity; these entries form $\mathcal{E} = \{m_j\}_{j=1}^J$. Full construction and retrieval details are provided in Appendix A.4.

Raw Experience as a Minimal Substrate

Our experience bank and similarity retriever are intentionally simple. Richer sources such as off-the-shelf agent skills, reusable codebooks, or learned retrievers can be incorporated without changing the framework. The point is not to prescribe another experience format, but to show that useful privileged context can be learned directly from minimally processed trajectories rather than hand-crafted rules.

Given x and the retrieved set $\mathcal{E}$, the composer first encodes each item into hidden states:

$$\mathbf{H}_j = \mathrm{Enc}_\psi(x, m_j) \in \mathbb{R}^{T_j \times d}, \tag{5}$$

where m_j is the j-th retrieved trajectory, Enc_ψ is the encoder that maps the task–experience pair into hidden states, T_j is the encoded sequence length, and d is the hidden dimension. The variable-length states are compressed into a fixed number of latent tokens by a lightweight latent compressor. In

our implementation, this compressor is QFormer-style cross-attention with learned queries, although other latent-token generators can be used under the same abstraction:

$$\mathbf{E}_j = \mathrm{Comp}_\chi(\mathbf{H}_j; \mathbf{Q}_\chi) = [\langle e_{j,1}\rangle, \ldots, \langle e_{j,K}\rangle] \in \mathbb{R}^{K\times d}, \tag{6}$$

where Comp_χ is the compressor, $\mathbf{Q}_\chi \in \mathbb{R}^{K\times d}$ is a learned query bank, and $\mathbf{E}_j$ is the latent representation of experience item m_j. We use $\phi := (\psi, \chi)$ to denote all trainable composer parameters, comprising the encoder LoRA parameters ψ and compressor parameters χ; the encoder backbone itself remains frozen. Note that we do not treat the particular attention block as a contribution; its role is simply to produce a compact, differentiable context (details in Appendix A.2). The teacher receives the latent tokens as ordinary context positions:

$$\boldsymbol{c}_\phi(x, \mathcal{E}) = \bigoplus_{j=1}^{J} (\langle e_{j,1}\rangle \oplus \cdots \oplus \langle e_{j,K}\rangle), \qquad \pi^T_{\bar\theta,\phi}(\cdot \mid \boldsymbol{s}, \boldsymbol{c}_\phi) = \pi_{\bar\theta}(\cdot \mid [\boldsymbol{s}; \boldsymbol{c}_\phi]), \tag{7}$$

where $\boldsymbol{s}$ is an interaction state defined above and $\boldsymbol{c}_\phi$ is the learned privileged context. Each $\langle e_{j,k}\rangle$ is implemented as a continuous embedding and behaves like a special token in the teacher's context window. The composer is cold-started on successful trajectories with the backbone frozen, yielding ϕ_0. During subsequent joint optimization, both the encoder LoRA parameters ψ and compressor parameters χ remain trainable as part of ϕ, while the teacher backbone $\bar\theta$ stays fixed (details in Appendix A.3).

3.3 Latent On-Policy Self-Distillation

Having specified how the teacher is constructed, we now describe the self-distillation process. The student first produces a multi-turn trajectory conditioned only on $\boldsymbol{s}_t$. The composer then constructs $\boldsymbol{c}_\phi$ from retrieved experience, and the teacher re-evaluates the same student prefixes with this additional context. We instantiate the teacher as a frozen reference copy $\bar\theta$ initialized from the same backbone as the student; the two share architecture and initialization, but not weights after the student begins updating. For every turn t and token position n, we define:

$$\boldsymbol{p}^S_{t,n} = \pi^S_\theta\big(\cdot \mid \boldsymbol{s}_t, a_{t,<n}\big), \ \boldsymbol{p}^T_{t,n} = \pi^T_{\bar\theta,\phi}\big(\cdot \mid \boldsymbol{s}_t, \boldsymbol{c}_\phi(x, \mathcal{E}), a_{t,<n}\big), \tag{8}$$

where $\boldsymbol{p}^S_{t,n}$ and $\boldsymbol{p}^T_{t,n}$ are next-token distributions over the vocabulary $\mathcal{V}$, and $\bar\theta$ denotes a fixed base-model checkpoint whose parameters receive no gradient updates; however, the teacher's forward activations remain in the computation graph, so $\boldsymbol{p}^T_{t,n}$ is differentiable with respect to $\boldsymbol{c}_\phi$ and hence to ϕ. To keep logit-level distillation efficient, we follow the truncated logit distillation of SDPO (Hübotter et al., 2026), keeping the teacher's top-M vocabulary entries plus a tail bucket. This preserves the teacher's high-probability modes while accounting for the remaining probability mass:

$$\tilde{\boldsymbol{p}}^T_{t,n}(v) = \begin{cases} \boldsymbol{p}^T_{t,n}(v), & v \in \mathcal{V}^M_{t,n}, \\ 1 - \sum_{u\in\mathcal{V}^M_{t,n}} \boldsymbol{p}^T_{t,n}(u), & v = \perp, \end{cases} \ \tilde{\boldsymbol{p}}^S_{t,n}(v) = \begin{cases} \boldsymbol{p}^S_{t,n}(v), & v \in \mathcal{V}^M_{t,n}, \\ 1 - \sum_{u\in\mathcal{V}^M_{t,n}} \boldsymbol{p}^S_{t,n}(u), & v = \perp, \end{cases} \tag{9}$$

where $\mathcal{V}^M_{t,n}$ is the teacher top-M support at prefix (t, n) and $\perp$ denotes the aggregated tail event. The distillation objective matches teacher and student on the student's own multi-turn trajectory:

$$\mathcal{L}_{\text{distill}}(\theta, \phi) = \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{\boldsymbol{\tau}\sim\pi^S_\theta(\cdot|x)}\left[\frac{\sum_{t=1}^{|\boldsymbol{\tau}|}\sum_{n=1}^{L_t} \omega_{t,n}\, D_{\mathrm{KL}}\big(\tilde{\boldsymbol{p}}^S_{t,n} \big\| \tilde{\boldsymbol{p}}^T_{t,n}\big)}{\sum_{t=1}^{|\boldsymbol{\tau}|}\sum_{n=1}^{L_t}\omega_{t,n}}\right], \tag{10}$$

where $\mathcal{D}$ is the task distribution and $\omega_{t,n} \in \{0, 1\}$ masks supervised action tokens. We use reverse KL so that the student concentrates on teacher-supported behavior. Because $\bar\theta$ denotes a fixed checkpoint while ϕ remains trainable, the distillation gradient flows through the frozen network back to the latent input (injection mechanism in Appendix A.5). This allows the composer to learn which experience features produce effective teacher supervision for the current student trajectory.

Privileged-Margin Constraint. The distillation loss alone does not ensure that the teacher provides more effective supervision: an unconstrained composer can instead minimize Equation (10) by moving $\boldsymbol{p}^T$ toward $\boldsymbol{p}^S$, yielding uninformative latent context without improving the student. We therefore prevent this collapse in two ways. ❶ First, the cold start in Section 3.2 initializes the composer to transform experience into informative latent context. ❷ Second, we introduce outcome

reward into the supervision signal, and more concretely, impose a privileged-margin constraint that ties the teacher's token-level advantage to the verified outcome of the complete trajectory. For each supervised token, we define the per-token privilege:

$$\delta_{t,n}(\phi) = \log \pi^T_{\bar{\theta},\phi}(a_{t,n} \mid \boldsymbol{s}_t, \boldsymbol{c}_\phi, a_{t,<n}) - \mathrm{sg}\big[\log \pi^S_\theta(a_{t,n} \mid \boldsymbol{s}_t, a_{t,<n})\big], \tag{11}$$

where $a_{t,n}$ is the student's sampled token and sg blocks gradients to θ through this path. Both log-probabilities are already computed during the teacher and student forward passes; $\delta_{t,n}$ requires only a gather, not an additional forward. We then use the trajectory-level verification signal $A(\boldsymbol{\tau}) = 2r(\boldsymbol{\tau}) - 1 \in [-1, 1]$, where $r(\boldsymbol{\tau})$ is the outcome reward, to favor teacher advantages on successful trajectories and suppress them on unsuccessful ones:

$$\Delta(\phi) = \mathbb{E}_{\boldsymbol{\tau}}\left[\frac{\sum_{t,n} \omega_{t,n}\, A(\boldsymbol{\tau})\, \delta_{t,n}(\phi)}{\sum_{t,n} \omega_{t,n}}\right]. \tag{12}$$

The full LOPD objective constrains ϕ to maintain a minimum privilege level $m > 0$:

$$\min_{\theta,\phi} \max_{\beta \geq 0} \; \mathcal{L}_{\text{distill}}(\theta, \phi) \; + \; \beta(m - \Delta(\phi)) + \; \lambda \left\| \boldsymbol{c}_\phi - \mathrm{sg}[\boldsymbol{c}_{\phi_0}] \right\|_2^2, \tag{13}$$

where β is a dual variable updated by $\beta \leftarrow [\beta + \eta_\beta(m - \Delta(\phi))]_+$, and the anchor term, taken as the mean squared deviation over latent entries, penalizes drift from the initialization ϕ_0 in latent space, requiring no additional forward pass. If the composer degenerates to uninformative context, $\pi^T \to \pi^S$, $\delta_{t,n} \to 0$, $\Delta \to 0 < m$, and the dual penalty activates—structurally excluding the trivial solution. Within the feasible region, outcome-weighted privilege steers the composer toward evidence that supports successful behavior, while the distillation gradient determines how that evidence is selected and encoded. The complete training procedure is summarized in Algorithm 1.

4 Experiments

4.1 Experiment Setup

Training. We evaluate LOPD under two post-training settings: ❶ **agentic tool-use** and ❷ **coding**. For tool-use training, we use the EnvScaler-derived tool-interactive corpus (Song et al., 2026), comprising 2,349 tasks. For coding, we use the TACO subset of DeepCoder (TogetherAI, 2025), comprising 7K verified Python problems (see Appendix B.2 for training details). The used model backbones include Qwen3-4B, Qwen3-8B (Yang et al., 2025), Olmo3-7B-Think (Olmo et al., 2026).

Evaluation. Our evaluation mirrors the two training regimes while keeping all test sets disjoint from training data. For tool-use, we report the EnvScaler task success metric on a held-out test split of 200 tasks disjoint from the training pool, BFCL-v3 scores across base, missing-function, missing-parameter, and long-context subsets (Patil et al., 2025), and ACEBench multi-step and multi-turn scores (Chen et al., 2025). For coding, we report pass@1 on LiveCodeBench v5/v6 (Jain et al., 2024), HumanEval+ and MBPP+ (Liu et al., 2023) (evaluation protocols in Appendix B.3).

Baselines. We compare against the baselines reported in Tables 1 and 2. **Vanilla** denotes the unadapted backbone. **GRPO** is the outcome-reward RL baseline (Shao et al., 2024). **SDFT** (Shenfeld et al., 2026) is a demonstration-conditioned distillation baseline. The OPSD-style baselines include **OPSD** (Zhao et al., 2026), **SDPO** (Hübotter et al., 2026), and **Skill-SD** (Wang et al., 2026), whose privileged contexts respectively instantiate answer/trace, feedback-conditioned self-teacher, and skill-conditioned supervision. Detailed baseline configurations are provided in Appendix B.1.

Configurations. All trainable methods share the same backbone, training split and evaluation protocol within each setting. For LOPD, the teacher receives latent context composed from J=3 retrieved experiences, each compressed into K=32 latent tokens (96 total); the distillation loss uses reverse KL with SDPO-style top-M logits plus a tail bucket, with M=20 by default. The privileged-margin threshold is m=0.05; the verification signal is $A(\boldsymbol{\tau}) = 2r(\boldsymbol{\tau}) - 1$ where r is the environment reward. Tool-use rollouts are capped at 30 environment steps. Coding distillation uses a 16,384-token response budget. We keep decoding temperature/top-p fixed across methods within each benchmark, each baseline retains its original KL direction and loss formulation (Appendix B.1), and within the on-policy training loop the methods differ mainly in privileged context construction and distillation loss. Prompt templates are provided in Appendix A.6.

Table 1: Tool-use results across EnvScaler, BFCL-v3, and ACEBench. All results are reported on a 0–100 scale; higher is better. Green and light-green cells mark the best and second-best results.

LLM	Method	EnvScaler	BFCL-v3					ACEBench		
			Base	Miss Func	Miss Param	Long Ctx	Avg	M-Step	M-Turn	Avg
QWEN3-4B	Vanilla	48.6	28.50	20.50	20.50	22.00	22.88	48.3	52.2	50.6
	GRPO	61.8	**33.00**	20.50	**24.50**	23.00	25.25	**58.3**	54.4	56.0
	Skill-SD	59.1	**33.00**	19.00	24.00	22.50	24.63	56.6	55.6	56.0
	OPSD	51.2	31.50	19.50	20.50	**29.00**	25.13	41.6	53.3	48.6
	SDPO	50.6	21.50	14.00	13.50	14.00	15.75	45.0	33.3	38.0
	SDFT	50.3	31.50	17.00	17.00	21.50	21.75	53.3	47.8	50.0
	LOPD	**63.7**	32.50	**26.50**	**24.50**	26.00	**27.38**	56.6	**63.3**	**60.6**
QWEN3-8B	Vanilla	49.2	34.00	28.50	25.50	25.50	28.38	61.6	50.0	54.6
	GRPO	57.3	36.00	28.00	27.00	25.00	29.00	65.0	53.3	58.0
	Skill-SD	60.2	33.00	27.00	25.00	24.50	27.38	63.3	51.1	56.0
	OPSD	52.0	30.50	25.00	23.50	24.00	25.75	60.0	47.8	52.7
	SDPO	55.3	29.00	24.50	23.00	23.50	25.00	56.7	48.9	52.0
	SDFT	56.2	31.50	26.00	24.50	25.50	26.88	65.0	47.8	54.7
	LOPD	**66.4**	**37.00**	**29.00**	**27.50**	**26.00**	**29.88**	**73.3**	**55.6**	**62.7**

4.2 MAIN RESULTS

We evaluate whether latent privileged context improves self-distillation learning across tool use and code generation domains in Tables 1 and 2.

No Hand-Crafted Context Is Universally Optimal. A careful reader may notice that adding privileged information does not always improve upon the vanilla model in Tables 1 and 2. For example, with QWEN3-4B, SDPO falls from 22.88 to 15.75 on BFCL-v3, from 50.6 to 38.0 on ACEBench, and from 45.61 to 38.78 on LiveCodeBench; OPSD likewise reduces the LiveCodeBench aggregate to 40.24. More revealingly, the utility of the same context can reverse across settings: OPSD raises the BFCL-v3 long-context score from 22.00 to 29.00 with QWEN3-4B, yet remains below vanilla on the overall BFCL-v3 score with QWEN3-8B (25.75 vs. 28.38) and on LiveCodeBench with both backbones (40.24 vs. 45.61 and 44.39 vs. 46.34). This behavior is consistent with how these contexts are constructed. SDPO conditions on successful siblings and environment feedback produced by the current rollout group, making its privileged signal dependent on what the current policy itself attempts; OPSD instead injects a fixed oracle trace, which can be highly informative when its procedure matches the current task but need not align with the particular prefix or valid solution path visited by the student. The issue is not simply whether privileged information is available, but whether its representation remains appropriate across tasks and learning states.

Table 2: Code results across LiveCodeBench and EvalPlus. Each cell reports pass@1 (%); higher is better. Green and light-green cells mark the best and second-best within each backbone group.

LLM	Method	LiveCodeBench			EvalPlus		
		v5	v6	Avg	HumanEval+	MBPP+	Avg
QWEN3-4B	Vanilla	47.67	41.22	45.61	85.37	75.93	78.79
	GRPO	50.18	44.27	48.29	86.59	76.19	79.34
	Skill-SD	49.82	41.98	47.32	85.98	76.98	79.70
	OPSD	42.65	35.11	40.24	85.36	76.72	79.33
	SDPO	40.86	34.35	38.78	84.76	74.87	77.86
	SDFT	48.75	43.51	47.07	87.20	76.98	80.07
	LOPD	**50.54**	**45.04**	**48.78**	**87.80**	**78.57**	**81.36**
OLMO3-7B	Vanilla	47.31	44.27	46.34	86.59	70.37	75.28
	GRPO	49.46	**45.80**	48.29	89.02	72.75	77.67
	Skill-SD	49.10	45.03	47.80	87.20	72.22	76.75
	OPSD	44.09	45.04	44.39	87.80	71.96	76.75
	SDPO	49.82	43.51	47.80	88.41	72.75	77.49
	SDFT	47.67	44.27	46.58	89.02	73.02	77.86
	LOPD	**53.41**	**45.80**	**50.98**	**90.24**	**73.28**	**78.41**

Learnable Context Matters. Can learning the context itself make privileged supervision more reliable across settings? LOPD learns a compact continuous representation directly from retrieved trajectories, allowing the privileged signal to adapt to the task and the student's visited prefixes. Unlike the reversals above, LOPD obtains the best aggregate result in every backbone–benchmark setting, with gains over vanilla ranging from 1.50 points on BFCL-v3 with QWEN3-8B to 17.2 points on EnvScaler with the same backbone. The advantage also holds against the strongest competing methods:

Metric	Vanilla	Base + Composer	LOPD
Reward	0.486	0.631	**0.637**
Interaction steps	11.12	16.31	17.04
Tool calls / step	3.50	1.21	**1.11**
First-step length	9,937	6,695	**6,210**
Reward / tool call	0.038	**0.053**	0.050
Repeated tool calls	8.89	**4.49**	5.25

Table 3: **Fine-grained behavior on the EnvScaler test set.** Base + Composer denotes the unadapted backbone conditioned on composed latent context; the student receives no privileged context.

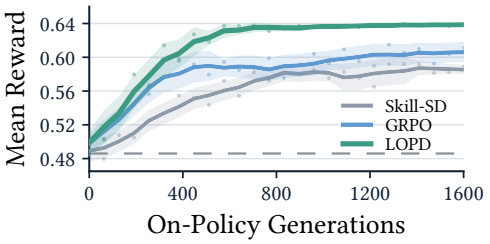

Figure 4: **EnvScaler training dynamics.** Mean reward over 1,600 rollouts.

with QWEN3-8B, LOPD reaches 66.4/29.88/62.7 on EnvScaler, BFCL-v3, and ACEBench, compared with the strongest baseline results of 60.2/29.00/58.0, respectively. The same trend extends beyond interactive agents. With QWEN3-4B, LOPD improves the LiveCodeBench and EvalPlus aggregates to 48.78 and 81.36; with OLMO3-7B, it reaches 50.98 and 78.41, outperforming the strongest alternatives by 2.69 and 0.55 points. Together, the results favor optimizing the representation of experience for supervision over searching for an increasingly elaborate hand-crafted artifact.

4.3 FRAMEWORK ANALYSIS

Effect of Joint Optimization. Does joint optimization actually produce a better privileged context, or merely introduce additional trainable parameters? This is the central ablation behind our claim. Figure 3 compares a frozen composer with jointly optimized variants and evaluates each resulting student without retrieved experience or latent context, so the difference reflects what context learning transfers into the policy. Training with a frozen composer (ϕ_0) achieves 0.573. Without the margin constraint (m=0), the student drops to 0.551, indicating that unconstrained distillation gradients degrade the latent context. Weak margins ($m \leq 0.01$) do not prevent this decline. With $m \geq 0.02$, the student surpasses the frozen-composer baseline, reaching 0.637 at m=0.05 and 0.626 at m=0.10. The improvement suggests that the distillation process exposes the composer to a signal unavailable during initialization: what evidence the current student's trajectory distribution requires from the privileged context. The margin constraint is necessary to realize this benefit—without it, collapsing the teacher toward the student is a lower-resistance path than learning a more informative representation.

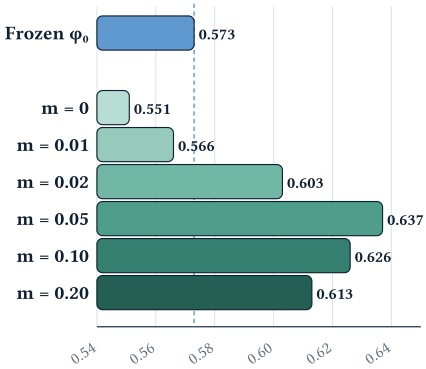

Figure 3: **Effect of joint optimization.** EnvScaler performance across margins.

Training Dynamics and Sample Efficiency. Beyond final performance, a practical question is whether LOPD reaches strong performance with fewer on-policy generations. To test this, Figure 4 tracks the strongest baselines over the same 1,600-generation budget. LOPD exceeds 0.61 mean reward after 320 generations and reaches 0.637 by generation 576. Its reward then remains in a narrow 0.63–0.64 range through generation 1,600, showing that the early gain is sustained rather than caused by a shorter training horizon. GRPO and Skill-SD improve more gradually and finish at 0.611 and 0.588, respectively. The persistent early separation suggests that the latent teacher extracts a denser learning signal from each visited trajectory.

Sensitivity Analysis. How much latent capacity and retrieved experience does the learnable substrate actually need? We vary the number of latent tokens produced by the composer and the number of experiences retrieved during training, then evaluate the resulting student alone. Figure 5(a) shows a capacity threshold in the latent bottleneck. EnvScaler reward remains near 0.56 with 8 or 16 tokens per experience, rises sharply to 0.637 with 32, and then fluctuates without a consistent gain at 64 and 128. We therefore use K=32 tokens per experience as the smallest setting that escapes the low-capacity regime. Retrieval sensitivity is shown in Figure 5(b–e), where n_{ret} varies from 1 to 10. EnvScaler reward improves from 0.605 with one retrieval to 0.637 with three, but additional retrievals yield no monotonic benefit. At the default n_{ret}=3, ACEBench reaches 56.6 on M-Step, 63.3 on M-Turn, and 60.6 overall, matching the main result in Table 1. Larger retrieval counts can

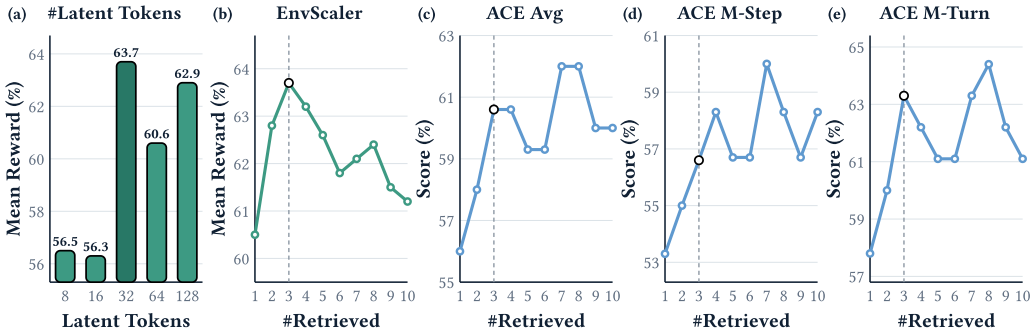

Figure 5: **Sensitivity analysis.** Performance of the resulting student without privileged context. (a) Latent-token capacity. (b) EnvScaler reward and (c–e) ACEBench aggregate, multi-step, and multi-turn scores as the training-time retrieval count varies. Dashed lines mark the default setting.

improve one ACEBench regime without producing the same trajectory in the other, and the aggregate remains on a broad plateau rather than varying monotonically. We therefore retain $n_{\text{ret}}=3$ as the earliest setting that attains the strongest EnvScaler reward while already reaching competitive ACEBench performance.

Behavioral Internalization. Finally, higher reward alone does not reveal what the student has internalized. We therefore ask whether LOPD changes how the student interacts with the environment. Table 3 shows that the distilled student inherits the interaction pattern induced by latent context. Relative to the vanilla model, LOPD uses more environment steps (17.04 vs. 11.12) but makes far fewer tool calls per step (1.11 vs. 3.50), indicating a shift from issuing many speculative calls at once to executing a more sequential plan. Its first-step response is 37.5% shorter, repeated calls fall from 8.89 to 5.25, and reward per tool call rises from 0.038 to 0.050. The latent-context-conditioned base model and the final student exhibit the same qualitative pattern, providing behavioral evidence that LOPD internalizes the teacher's procedural guidance rather than merely fitting the aggregate reward. Per-experiment configurations are detailed in Appendix B.4.

4.4 Case Study

Having established that latent context improves learning, a natural question is what these continuous tokens encode. We project all 32 latent tokens through the frozen language-model head and inspect ten tool-use and ten coding examples, with and without task conditioning. Figure 6 in Appendix C shows two representative cases: the agentic task and retrieved trajectory share an add–update–change–withdraw schema across different entities, while the coding pair shares a circular-elimination structure captured by the retrieved Josephus recurrence. Yet both projections remain fragmented mixtures of multilingual and code-like tokens; task conditioning changes their surface form without producing readable procedures or copying the retrieved solutions. This is consistent with a distributed representation, although the projections alone do not establish which information the teacher functionally uses.

5 Conclusion

The question behind this work is whether the teacher's privileged context can itself be learned from experience. LOPD answers it by transforming retrieved raw trajectories into differentiable latent context, supervising the student's visited prefixes with the resulting self-teacher, and preserving a verifiable teacher advantage through a privileged-margin constraint. Across tool use and code generation, LOPD achieves the best aggregate result and surpasses GRPO and Skill-SD with less than 30% of their rollout budget. The analyses sharpen this interpretation: a frozen composer is insufficient, unconstrained joint optimization can collapse the teacher toward the student, and the privileged margin makes context learning productive. The resulting behavioral shift persists in the student without privileged context at inference. More broadly, scalable self-evolution calls for learning what makes experience useful, with raw trajectories as a minimal substrate and richer repositories or retrievers as possible extensions. The central design principle is to optimize experience representations end-to-end for the policies they are meant to improve.

AI USE STATEMENT

We used OpenAI Codex and Anthropic Claude to assist with literature research, manuscript polishing, figure preparation, and code development. The use of language models to generate training trajectories and support evaluation simulations is described in Appendices A.3, B.1 and B.3. We have reviewed all AI-assisted work and take responsibility for the final content of this paper, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

We have not identified prominent ethical risks specific to LOPD beyond those generally associated with large language models and tool-using agents. These include unreliable or biased outputs and the potential misuse of generated code or tool actions.

REPRODUCIBILITY STATEMENT

To support reproducibility, we provide an anonymous GitHub repository at https://anonymous.4open.science/r/lopd-code-2E63. Algorithm 1 provides pseudocode for the complete training procedure. Appendix A documents the composer architecture, cold-start initialization, retrieval configuration, latent injection mechanism, and prompt formats. Appendix B describes the baseline configurations, training data, evaluation protocols, and ablation settings used in our experiments.

REFERENCES

Ido Amos, Avi Caciularu, Mor Geva, Amir Globerson, Jonathan Herzig, Lior Shani, and Idan Szpektor. Latent reasoning with supervised thinking states, 2026. URL https://arxiv.org/abs/2602.08332.

Di Cao, Dongjie Fu, Hai Yu, Siqi Zheng, Xu Tan, and Tao Jin. X-OPD: Cross-Modal On-Policy Distillation for Capability Alignment in Speech LLMs. *arXiv preprint arXiv:2603.24596*, 2026. URL https://arxiv.org/abs/2603.24596.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Yuefeng Huang, Wulong Liu, Xinzhi Wang, Defu Lian, Baoqun Yin, Yasheng Wang, and Wu Liu. Acebench: Who wins the match point in tool usage?, 2025. URL https://arxiv.org/abs/2501.12851.

Jingcheng Deng, Liang Pang, Zihao Wei, Shicheng Xu, Zenghao Duan, Kun Xu, Yang Song, Huawei Shen, and Xueqi Cheng. Llm latent reasoning as chain of superposition, 2026. URL https://arxiv.org/abs/2510.15522.

Tao Feng, Chongrui Ye, Tianyang Luo, Jingjun Xu, Xueqiang Xu, Haozhen Zhang, Ge Liu, and Jiaxuan You. Elasticmem: Latent memory as a learnable resource for llm agents, 2026. URL https://arxiv.org/abs/2605.30690.

Yuqian Fu, Haohuan Huang, Kaiwen Jiang, Jiacai Liu, Zhuo Jiang, Yuanheng Zhu, and Dongbin Zhao. Revisiting On-Policy Distillation: Empirical Failure Modes and Simple Fixes. *arXiv preprint arXiv:2603.25562*, 2026. URL https://arxiv.org/abs/2603.25562.

Michael Hanna and Emmanuel Ameisen. Latent planning emerges with scale, 2026. URL https://arxiv.org/abs/2604.12493.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space, 2025. URL https://arxiv.org/abs/2412.06769.

Yinghui He, Simran Kaur, Adithya Bhaskar, Yongjin Yang, Jiarui Liu, Narutatsu Ri, Liam Fowl, Abhishek Panigrahi, Danqi Chen, and Sanjeev Arora. Self-Distillation Zero: Self-Revision Turns Binary Rewards into Dense Supervision. *arXiv preprint arXiv:2604.12002*, 2026. URL https://arxiv.org/abs/2604.12002.

Yubo Hou, Zhisheng Chen, Tao Wan, and Zengchang Qin. Flashmem: Distilling intrinsic latent memory via computation reuse, 2026. URL https://arxiv.org/abs/2601.05505.

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, 2026. URL https://arxiv.org/abs/2601.20802.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024. URL https://arxiv.org/abs/2403.07974.

Woogyeol Jin, Taywon Min, Yongjin Yang, Swanand Ravindra Kadhe, Yi Zhou, Dennis Wei, Nathalie Baracaldo, and Kimin Lee. Entropy-Aware On-Policy Distillation of Language Models. *arXiv preprint arXiv:2603.07079*, 2026. URL https://arxiv.org/abs/2603.07079.

Aristotelis Lazaridis, Dylan Bates, Aman Sharma, Brian King, Vincent Lu, and Jack FitzGerald. Edge-opd: Internalizing privileged context with evidence guided on-policy distillation, 2026. URL https://arxiv.org/abs/2605.23493.

Jiaze Li, Hao Yin, Haoran Xu, Boshen Xu, Wenhui Tan, Zewen He, Jianzhong Ju, Zhenbo Luo, and Jian Luan. Video-OPD: Efficient Post-Training of Multimodal Large Language Models for Temporal Video Grounding via On-Policy Distillation. *arXiv preprint arXiv:2602.02994*, 2026a. URL https://arxiv.org/abs/2602.02994.

Yaxuan Li, Yuxin Zuo, Bingxiang He, Jinqian Zhang, Chaojun Xiao, Cheng Qian, Tianyu Yu, Huan ang Gao, Wenkai Yang, Zhiyuan Liu, and Ning Ding. Rethinking on-policy distillation of large language models: Phenomenology, mechanism, and recipe, 2026b. URL https://arxiv.org/abs/2604.13016.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation, 2023. URL https://arxiv.org/abs/2305.01210.

Ruiqi Liu, Xiaolei Lv, Gengsheng Li, Ximo Zhu, Zhiheng Wang, Zhengbo Zhang, Junkai Chen, Zhiheng Li, Bo Li, Jun Gao, and Shu Wu. Visual-Advantage On-Policy Distillation for Vision-Language Models. *arXiv preprint arXiv:2605.21924*, 2026. URL https://arxiv.org/abs/2605.21924.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3, 2026. URL https://arxiv.org/abs/2512.13961.

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, Fanjia Yan, Charlie Cheng-Jie Ji, 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://openreview.net/forum?id=2GmDdhBdDk.

Emiliano Penaloza, Dheeraj Vattikonda, Nicolas Gontier, Alexandre Lacoste, Laurent Charlin, and Massimo Caccia. Privileged information distillation for language models, 2026. URL https://arxiv.org/abs/2602.04942.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning, 2026. URL https://arxiv.org/abs/2601.19897.

Mingyang Song and Mao Zheng. A survey of on-policy distillation for large language models, 2026. URL https://arxiv.org/abs/2604.00626.

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Envscaler: Scaling tool-interactive environments for llm agent via programmatic synthesis, 2026. URL https://arxiv.org/abs/2601.05808.

TogetherAI. DeepCoder: A Fully Open-Source 14B Coder at O3-mini Level. https://www.together.ai/blog/deepcoder, 2025.

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, 2026. URL https://arxiv.org/abs/2604.10674.

Yu Wang, Yifan Gao, Xiusi Chen, Haoming Jiang, Shiyang Li, Jingfeng Yang, Qingyu Yin, Zheng Li, Xian Li, Bing Yin, Jingbo Shang, and Julian McAuley. Memoryllm: Towards self-updatable large language models, 2024. URL https://arxiv.org/abs/2402.04624.

Yecheng Wu, Song Han, and Han Cai. Lightning OPD: Efficient Post-Training for Large Reasoning Models with Offline On-Policy Distillation. *arXiv preprint arXiv:2604.13010*, 2026a. URL https://arxiv.org/abs/2604.13010.

Zijun Wu, Yongchang Hao, and Lili Mou. Tokmem: One-token procedural memory for large language models, 2026b. URL https://arxiv.org/abs/2510.00444.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Chenxu Yang, Chuanyu Qin, Qingyi Si, Minghui Chen, Naibin Gu, Dingyu Yao, Zheng Lin, Weiping Wang, Jiaqi Wang, and Nan Duan. Self-distilled rlvr, 2026. URL https://arxiv.org/abs/2604.03128.

Tianzhu Ye, Li Dong, Xun Wu, Shaohan Huang, and Furu Wei. On-policy context distillation for language models, 2026. URL https://arxiv.org/abs/2602.12275.

Weichen Yu, Xiaomin Li, Yizhou Zhao, Xiaoze Liu, Ruowang Zhang, Haixin Wang, Yinyi Luo, Chen Henry Wu, Gaurav Mittal, Matt Fredrikson, and Yu Hu. Multi-Rollout On-Policy Distillation via Peer Successes and Failures. *arXiv preprint arXiv:2605.12652*, 2026a. URL https://arxiv.org/abs/2605.12652.

Xinlei Yu, Zhangquan Chen, Yongbo He, Tianyu Fu, Guanting Dong, Cheng Yang, Chengming Xu, Yue Ma, Xiaobin Hu, Zhe Cao, Jie Xu, Guibin Zhang, Jiale Tao, Jiayi Zhang, Siyuan Ma, Kaituo Feng, Haojie Huang, Youxing Li, Ronghao Chen, Huacan Wang, Chenglin Wu, Zikun Su, Xiaogang Xu, Kelu Yao, Kun Wang, Chen Gao, Yue Liao, Ruqi Huang, Tao Jin, Zhucun Xue, Cheng Tan, Jiangning Zhang, Wenqi Ren, Yanwei Fu, Yong Liu, Yu Wang, Xiangyu Yue, Yu-Gang Jiang, and Shuicheng Yan. The latent space: Foundation, evolution, mechanism, ability, and outlook, 2026b. URL https://arxiv.org/abs/2604.02029.

Xinlei Yu, Gen Li, Qingyi Si, Guibin Zhang, Yuqi Xu, Congcong Wang, Shuai Dong, Kaiwen Tuo, Xiangyu Zeng, Kaituo Feng, et al. Dopd: Dual on-policy distillation. *arXiv preprint arXiv:2606.30626*, 2026c. URL https://arxiv.org/abs/2606.30626.

Xinlei Yu, Chengming Xu, Guibin Zhang, Zhangquan Chen, Yudong Zhang, Yongbo He, Peng-Tao Jiang, Jiangning Zhang, Xiaobin Hu, and Shuicheng Yan. Vismem: Latent vision memory unlocks potential of vision-language models, 2026d. URL https://arxiv.org/abs/2511.11007.

Qianhao Yuan, Jie Lou, Xing Yu, Hongyu Lin, Le Sun, Xianpei Han, and Yaojie Lu. Vision-OPD: Learning to See Fine Details for Multimodal LLMs via On-Policy Self-Distillation. *arXiv preprint arXiv:2605.18740*, 2026. URL https://arxiv.org/abs/2605.18740.

Guibin Zhang, Muxin Fu, and Shuicheng Yan. Memgen: Weaving generative latent memory for self-evolving agents, 2025. URL https://arxiv.org/abs/2509.24704.

Xinsen Zhang, Zhenkai Ding, Tianjun Pan, Run Yang, Chun Kang, Xue Xiong, and Jingnan Gu. OPSDL: On-Policy Self-Distillation for Long-Context Language Models. *arXiv preprint arXiv:2604.17535*, 2026a. URL https://arxiv.org/abs/2604.17535.

Yan Zhang, Daiqing Wu, Huawen Shen, Yu Zhou, and Can Ma. Learn where to Click from Yourself: On-Policy Self-Distillation for GUI Grounding. *arXiv preprint arXiv:2605.00642*, 2026b. URL https://arxiv.org/abs/2605.00642.

Yuwei Zhang, Sha Li, Changlong Yu, Qin Lu, Shuowei Jin, Chengyu Dong, Haoran Liu, Ilgee Hong, Xintong Li, Zhenyu Shi, Bing Yin, and Jingbo Shang. Learning with Rare Success but Rich Feedback via Reflection-Enhanced Self-Distillation. *arXiv preprint arXiv:2605.12741*, 2026c. URL https://arxiv.org/abs/2605.12741.

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, 2026. URL https://arxiv.org/abs/2601.18734.

Zhi Zheng and Wee Sun Lee. Beyond imitation: Reinforcement learning for active latent planning, 2026. URL https://arxiv.org/abs/2601.21598.

Rui-Jie Zhu, Tianhao Peng, Tianhao Cheng, Xingwei Qu, Jinfa Huang, Dawei Zhu, Hao Wang, Kaiwen Xue, Xuanliang Zhang, Yong Shan, Tianle Cai, Taylor Kergan, Assel Kembay, Andrew Smith, Chenghua Lin, Binh Nguyen, Yuqi Pan, Yuhong Chou, Zefan Cai, Zhenhe Wu, Yongchi Zhao, Tianyu Liu, Jian Yang, Wangchunshu Zhou, Chujie Zheng, Chongxuan Li, Yuyin Zhou, Zhoujun Li, Zhaoxiang Zhang, Jiaheng Liu, Ge Zhang, Wenhao Huang, and Jason Eshraghian. A survey on latent reasoning, 2025. URL https://arxiv.org/abs/2507.06203.

A Implementation Details

A.1 Training Algorithm

Algorithm 1: Latent On-Policy Self-Distillation (LOPD).

Input : Task distribution $\mathcal{D}$; retriever Ret; student π_θ^S; fixed teacher $\pi_{\bar{\theta}}^T$; composer Φ_ϕ (init. from ϕ_0); reward verifier $V : \boldsymbol{\tau} \mapsto [0, 1]$; margin m; dual step η_β

Output: Trained student policy π_θ^S

$\beta \leftarrow 0$; cache $\boldsymbol{c}_{\phi_0}$ per task
for each training iteration **do**
  Sample tasks $\{x_i\}$ from $\mathcal{D}$
  foreach task x_i **do**
    /* On-policy student rollout */
    Sample trajectory $\boldsymbol{\tau}_i \sim \pi_\theta^S(\cdot \mid x_i)$; $r_i \leftarrow V(\boldsymbol{\tau}_i)$; $A_i \leftarrow 2r_i - 1$
    /* Teacher evaluation with learned privileged context */
    $\mathcal{E}_i \leftarrow \mathrm{Ret}(x_i)$;
    $\boldsymbol{c}_{\phi,i} \leftarrow \Phi_\phi(x_i, \mathcal{E}_i)$
    foreach turn t and action prefix $a_{t,<n}$ in $\boldsymbol{\tau}_i$ **do**
      Compute $\boldsymbol{p}_{t,n}^S$ and $\boldsymbol{p}_{t,n}^T$ (Equation (8)); form top-M-plus-tail
      Accumulate $D_{\mathrm{KL}}(\tilde{\boldsymbol{p}}_{t,n}^S \| \tilde{\boldsymbol{p}}_{t,n}^T)$
      $\delta_{t,n} \leftarrow \log \boldsymbol{p}_{t,n}^T(a_{t,n}) - \mathrm{sg}[\log \boldsymbol{p}_{t,n}^S(a_{t,n})]$
  $\Delta \leftarrow$ weighted mean of $A_i \cdot \delta_{t,n}$ over masked tokens
  Update (θ, ϕ) by minimizing Equation (13)
  $\beta \leftarrow [\beta + \eta_\beta(m - \Delta)]_+$

A.2 Composer Architecture

The latent composer Φ_ϕ consists of an encoder Enc_ψ and a compressor Comp_χ, with $\phi := (\psi, \chi)$. The encoder uses the frozen backbone augmented with a trainable LoRA adapter (rank 8, α=16, dropout 0, applied to all seven linear projections per transformer block: $\mathbf{W}_q$, $\mathbf{W}_k$, $\mathbf{W}_v$, $\mathbf{W}_o$, gate, up, and down projections). During encoding, the task description is prepended to each retrieved experience to enable task-conditional compression, so that the QFormer's cross-attention can condition on the current task when compressing the experience (see Appendix A.6 for the exact format).

The compressor is a QFormer-style perceiver where learned queries cross-attend to the encoder's hidden states. It has 8 cross-attention layers with shared parameters (i.e., all layers reuse the same weights), attention heads and hidden dimension inherited from the backbone model, and a feed-forward multiplier of 4. Learned queries $\mathbf{Q}_\chi \in \mathbb{R}^{K \times d}$ are initialized from $\mathcal{N}(0, 1/\sqrt{d})$. The compressor operates independently on each retrieved experience, producing K latent tokens per item; the outputs are concatenated to form the full latent context $\boldsymbol{c}_\phi$. Based on the sensitivity analysis in Section 4.3, we set K=32 tokens per retrieved experience for the final LOPD training runs. The encoder LoRA and QFormer are updated during both cold-start and joint optimization; the backbone weights remain frozen throughout.

A.3 Cold-Start Initialization

The composer is cold-started by supervised finetuning on retrieved-experience–trajectory pairs with frozen backbone. The training objective is next-token NLL on assistant tokens with latent-augmented input:

$$\mathcal{L}_{\mathrm{init}}(\phi) = -\mathbb{E}_{(x, \boldsymbol{y}^\star) \sim \mathcal{D}_{\mathrm{exp}},\, \mathcal{E} = \mathrm{Ret}(x)} [\log \pi_{\bar{\theta}}(\boldsymbol{y}^\star \mid \boldsymbol{s}, \boldsymbol{c}_\phi(x, \mathcal{E}))] \,, \tag{14}$$

where $\boldsymbol{y}^\star$ is a successful trajectory and $\bar{\theta}$ denotes the frozen backbone. Concretely, the composer first produces latent tokens $\boldsymbol{c}_\phi$ from retrieved experiences and inserts them into the input embedding sequence at designated placeholder positions. The frozen backbone then performs a standard forward pass over this latent-augmented input. Although the backbone parameters receive no gradient updates, the loss gradient flows backward through the frozen layers' activations to the latent

token positions, and from there to the QFormer and LoRA parameters—the same mechanism as prefix-tuning.

The training data is synthesized from the base model's own rollouts—no external expert or stronger model is required—and filtered by task success. The data volume and number of training steps are adjusted per domain. Training uses AdamW with learning rate 10^{-5}, batch size 8, gradient clipping of 30.0 for the QFormer and 3.0 for the LoRA adapter, task-conditional compression, and n_{ret}=3 retrieved experiences per task. The LoRA adapter and QFormer are updated during cold-start and remain trainable during subsequent joint optimization; the backbone remains frozen.

A.4 Retrieval Configuration

Let $\mathcal{B} = \{(x_i, \boldsymbol{\tau}_i)\}_{i=1}^{|\mathcal{B}|}$ denote the experience bank, where each entry pairs a task description x_i with a successful trajectory $\boldsymbol{\tau}_i$. A dense encoder f_{ret} maps a task description to a unit-norm embedding. Both sides are encoded from the task description alone: each bank entry is indexed by its task description x_i, and a query is encoded from the current task x; the trajectory $\boldsymbol{\tau}_i$ is stored alongside and consumed only by the composer:

$$\boldsymbol{z}_i^{\text{doc}} = \frac{f_{\text{ret}}(x_i)}{\|f_{\text{ret}}(x_i)\|_2}, \qquad \boldsymbol{z}_q = \frac{f_{\text{ret}}(x)}{\|f_{\text{ret}}(x)\|_2}, \qquad \boldsymbol{z}_i^{\text{doc}},\, \boldsymbol{z}_q \in \mathbb{R}^{d_{\text{ret}}}. \tag{15}$$

Given a query task x, retrieval returns the n_{ret} entries with the highest cosine similarity:

$$\text{Ret}(x) = \text{top-n}_{\text{ret}} \left\{ (x_i, \boldsymbol{\tau}_i) \in \mathcal{B} \;\middle|\; \boldsymbol{z}_q{}^{\top} \boldsymbol{z}_i^{\text{doc}} \right\}. \tag{16}$$

We use Qwen3-Embedding-8B as f_{ret}, producing d_{ret}=4,096-dimensional embeddings. The index is implemented as a FAISS `IndexFlatIP` for exact inner-product search over the precomputed document embeddings. The bank $\mathcal{B}$ is constructed offline exclusively from successful rollouts generated on the training split: agentic trajectories must exceed a task-reward threshold, while coding trajectories must pass all test cases. No evaluation task, trajectory, or outcome is ever inserted into the bank, and the bank is frozen before evaluation. Consequently, LOPD receives neither evaluation-set information nor an additional task corpus; it uses only experience produced from the same training-task split available to the baselines. Trajectories are stored in an observation-lite format that omits verbose environment observations to reduce storage and encoding length. All query embeddings $\boldsymbol{z}_q$ are precomputed and cached per task, so the retriever does not need to be loaded during training or evaluation.

Each bank entry stores the task description followed by a linearized action–result trace in observation-lite format, as illustrated below:

Example experience-bank entry (agentic, truncated)

```
Task:
Acting on behalf of Alice Chan, update her health policy XJ-439220:  add Psychiatric Care
coverage ($3500 limit, $300 deductible), increase ER coverage limit to $7000, ...
Example steps (from a different task, NOT yet executed):
1.  get_policy_by_policy_number("XJ-439220") -> policy_id=POL-001
2.  get_exclusions_for_policy("POL-001") -> exclusion_id=EXCL-001, ...
3.  remove_exclusion_from_policy("POL-001", "EXCL-002") -> Success
4.  add_coverage_item_to_policy("POL-001", ...)  -> Coverage item added
5.  update_coverage_limit_or_deductible("COV-002", limit=7000) -> Updated
...
```

A.5 Latent Injection Mechanism

The latent tokens produced by the composer are injected into the backbone's input embeddings along two paths. During LOPD training, the student acts without latent context, so rollouts run on standard inference engines; the latent-conditioned teacher only scores the student's visited prefixes, which we implement as a single differentiable forward pass in PyTorch (HuggingFace Transformers) so that gradients reach the composer. For latent-conditioned generation, such as the Base + Composer evaluation in Table 3, prior latent-state work typically relies on HuggingFace Transformers or TRL, whose sampling throughput is far below that of modern inference engines; we instead implement latent injection on top of production inference engines (SGLang and vLLM). Both paths share the placeholder strategy below.

Placeholder strategy. A text-level sentinel string <|LATENT_PH|> is inserted into the first user message of the chat template, sandwiched between natural-language framing text (e.g., "*The following is a reference example...*" before and "*Now complete your task...*" after). The full prompt is then rendered via the tokenizer's chat template and split on the sentinel. Each half is tokenized independently, and $J \cdot K$ dummy token IDs (using the tokenizer's pad token) are inserted between them:

$$\text{input\_ids} = [\underbrace{t_1, \ldots, t_a}_{\text{before}}, \underbrace{t_{\text{pad}}, \ldots, t_{\text{pad}}}_{J \cdot K \text{ placeholders}}, \underbrace{t_{a+1}, \ldots, t_L}_{\text{after}}]. \tag{17}$$

This produces a contiguous span of placeholder positions at known absolute indices, without any tokenizer modification.

Embedding replacement. The full `input_ids` are embedded as usual, producing $\mathbf{X} \in \mathbb{R}^{L \times d}$. Before the first transformer layer, the placeholder embeddings are replaced with the latent tokens from the composer:

$$\mathbf{X}[p_k] \leftarrow \langle e_k \rangle, \quad k = 1, \ldots, J \cdot K. \tag{18}$$

In the training path, the replacement uses `torch.cat` over slices (rather than in-place assignment) so that autograd can track gradients through $\langle e_k \rangle$ back to the composer parameters. In both paths, the modified embedding tensor is then passed through the backbone's transformer layers without any architectural change; in the inference path, the operation is transparent to the engine's KV-cache, attention mask, and batching logic.

Engine-specific implementations. SGLang provides a `positional_embed_overrides` interface that accepts embeddings at specified absolute positions and performs the replacement in-GPU. vLLM provides a `prompt_embeds` interface: the full sequence is first embedded on CPU via a frozen copy of the embedding table, the latent positions are overwritten with the composer's output, and the complete embedding tensor is submitted to the engine. In both cases, the backbone model is unmodified and no gradients are required.

Equivalence verification. To confirm that the placeholder-based injection introduces no numerical artifacts, we replaced the latent span with known token embeddings from the vocabulary and compared the token-ID input path against the embedding-tensor input path. The two paths produce identical greedy-decoded sequences with negligible log-probability differences, verifying that latent injection is numerically equivalent to standard token-ID forwarding.

Encoder LoRA isolation. The encoder Enc_ψ uses a LoRA adapter to specialize hidden-state extraction for experience compression. This adapter is explicitly *disabled* during the main forward pass (both at inference and during teacher evaluation in training), so that the model behaves as a pure base actor conditioned on the injected latent context. This dual-use pattern—LoRA active for encoding, inactive for generation—avoids interference between the two roles.

A.6 Prompt and Interaction Format

Agentic system prompt. For tool-use tasks (EnvScaler, ACEBench), the agent receives a system prompt instructing it to complete tasks via step-by-step tool invocation:

Agentic system prompt (EnvScaler / ACEBench)

```
You are a helpful assistant.  When given a specific task, your goal is to complete it in an
interactive environment by making step-by-step use of available tools.
- Before completing the task, at each step, select a tool from the tool list and fill in
all required parameters, making sure that the values are valid.  Avoid making parallel tool
calls in one step.
- When you believe the task has been completed, respond only with 'Task Completed' to end
the trajectory, without adding any other content or making any tool calls.
- It is recommended to first call query tools to gather sufficient information, then use
modification tools to complete the task.  Adjust actions promptly based on the feedback
from the environment.
```

Coding system prompt. For TACO and LiveCodeBench, the model receives a minimal system instruction:

Coding system prompt (TACO / LiveCodeBench)

```
You are an expert Python programmer.  You will be given a question (problem specification)
and will generate a correct Python program that matches the specification and passes all
tests.
```

HumanEval+ and MBPP+ use no system prompt, following the official EvalPlus evaluation protocol. All coding benchmarks use Python as the target language.

Latent-context framing text. The latent privileged context is injected into the first user message, wrapped by domain-specific framing text. The framing instructs the model to treat the injected content as a reference from a different task, not as instructions to follow verbatim.

Agentic privileged-context framing (EnvScaler)

```
Before latent context:
The following is a REFERENCE EXAMPLE of how a DIFFERENT, already-solved task was handled,
shown only to illustrate the general approach.  It was performed in a SEPARATE session --
in YOUR task below, NOTHING has been done yet and the environment is in its initial state.
You must perform every step yourself by calling the tools and reading their actual results;
do NOT assume any step is already complete.
After latent context:
--- end of reference example ---
Now complete YOUR task below, starting from scratch (the environment is untouched):
```

Coding privileged-context framing (TACO)

```
Before:
The following is a reference from a similar programming problem.  Study the approach and
algorithm, then solve YOUR problem below.
After:
--- end of reference ---
```

The latent tokens produced by the composer replace the framing's interior (as described in Appendix A.5).

Encoder input format. When the encoder Enc_ψ processes a retrieved experience for compression, the input is formatted as "`Task to solve:\n`x`\n\nReference past trajectory:\n`m_j", where x is the current task description and m_j is the retrieved trajectory. This task-conditional formatting allows the QFormer to attend to task-relevant features when producing the latent tokens.

B Experiment Details

B.1 Baseline Setup

All trainable methods share the same base model, training task distribution, on-policy rollout budget (32 rollouts per step), optimizer (AdamW, learning rate 10^{-5}, gradient clipping 1.0), and evaluation protocol. Each distillation-based method uses the KL direction prescribed by its original paper: OPSD and SDFT use forward KL over the full vocabulary, SDPO uses reverse KL with truncated logits and a tail bucket, and Skill-SD uses sampled-token reverse KL with importance weighting. SDFT and OPSD require ground-truth demonstrations as privileged context for the teacher. These are drawn from a shared pool of verified-successful trajectories: for agentic tasks, trajectories must exceed a reward threshold; for coding tasks, solutions must pass all test cases. The pool is first populated from the base model's own rollouts; when its coverage is insufficient, we synthesize additional ground-truth trajectories by querying stronger models (DeepSeek-V4-Pro and Qwen3.7-Max in our experiments). Below we describe the method-specific configurations.

Vanilla. The unadapted backbone model, evaluated without any post-training.

GRPO. Standard group relative policy optimization (Shao et al., 2024). Each step samples $G{=}4$ rollouts per task with stochastic decoding; group-normalized advantages weight a PPO-clip objec-

tive with $\epsilon_{\text{lo}}=\epsilon_{\text{hi}}=0.2$. No teacher or privileged context is used; the training signal comes entirely from environment rewards.

SDFT. Demonstration-conditioned distillation (Shenfeld et al., 2026). The teacher receives oracle trajectories from the shared pool as textual in-context demonstrations appended to the first user message. The teacher is an EMA shadow of the student ($\alpha_{\text{EMA}}=0.01$, synced every step). The distillation loss is forward KL over the full vocabulary, following the official implementation.

SDFT demonstration injection format

```
Appended to user message:
This is an example for a response to the question:
{demo_text}
Now answer with a response of your own, including the thinking process.
```

OPSD. On-policy self-distillation (Zhao et al., 2026). The teacher receives oracle trajectories from the shared pool directly as privileged context in the system message. The teacher is frozen throughout training. The distillation loss is forward KL over the full vocabulary, as prescribed by the original paper.

OPSD GT injection format

```
Appended to system message:
The following is a reference plan for the upcoming task.  Use it only as guidance for your
own reasoning; you must still interact with the environment to complete the task.
{gt_text}
After reading the reference solution above, make sure you truly understand the reasoning
behind each step.  Now, using your own words and independent reasoning, derive the same
final answer.
```

SDPO. Self-distillation policy optimization (Hübotter et al., 2026). The teacher is an EMA shadow of the student (update rate 0.05), conditioned on successful sibling trajectories sampled within the same training step ($G=4$ rollouts per task) and on environment feedback from the student's own attempt. Unlike SDFT and OPSD, SDPO does not use an external trajectory pool; it filters from its own rollouts using the same domain-specific success criterion (reward threshold for agentic tasks, full test-case pass for coding tasks).

Skill-SD. Skill-conditioned self-distillation (Wang et al., 2026). Each task selects one skill from a skill bank via UCB1 ($c=\sqrt{2}$). Skills are distilled from rollout trajectories into structured summaries and injected into the teacher's first user message:

Skill-SD skill injection format

```
Appended to user message:
Here is a skill summary from a similar past task that may help guide your approach:
**What works:** {success_analysis}
**What to avoid:** {mistake_analysis}
**Suggested workflow:** {golden_workflow}
Now solve the current task using your own reasoning.
```

The loss combines PPO-clip ($\epsilon_{\text{lo}}=0.2$, $\epsilon_{\text{hi}}=0.28$) with importance-weighted sampled-token reverse KL ($\lambda_{\text{sdl}}=0.001$), following the original paper.

B.2 Training Data and Domain Setup

All methods are trained exclusively on the two datasets listed in Table 4; no additional task dataset is used. In particular, the LOPD experience bank contains only rollouts from the corresponding training split and excludes every evaluation task and trajectory. The agentic training corpus (EnvScaler) covers diverse tool-interactive scenarios including insurance, logistics, e-commerce, and healthcare, with each task requiring multi-turn API interactions to complete. The coding training corpus (TACO subset of DeepCoder) consists of competitive programming problems with verified test cases, spanning algorithmic topics such as dynamic programming, graph traversal, and string manipulation. Thinking mode is enabled throughout training, as in evaluation (Appendix B.3). The environment reward differs by domain: EnvScaler returns a continuous reward in $[0, 1]$ reflecting

the fraction of subtasks completed, while TACO returns a binary reward (1 if all test cases pass, 0 otherwise).

Parameter	Agentic	Coding
Training data	EnvScaler	TACO (DeepCoder)
Training tasks	2,349	∼7,000
Total generation budget (tokens)	40,960	16,384
Max environment steps	30	— (single-turn)
Decoding temperature		0.7
top-p		0.95
top-k (sampling)		20
LOPD *constraint parameters*		
Composer learning rate (lr_ϕ)		10^{-5}
Anchor weight (λ)		0.2
Dual step size (η_β)		0.5

Table 4: Domain-specific training configurations.

B.3 Evaluation Protocols

Table 5 summarizes the inference configuration for each benchmark. All benchmarks use thinking mode enabled.

Benchmark	Tasks	Temp	top-p	top-k	Budget (tokens)	Max steps
EnvScaler	200	0.7	0.95	20	40,960	30
BFCL-v3	4×200	0.7	0.95	20	40,960	30
ACEBench	50	0.7	0.95	20	40,960	20
LiveCodeBench	279+131	0.7	0.95	20	16,384	—
HumanEval+	164	0.7	0.95	20	16,384	—
MBPP+	378	0.7	0.95	20	16,384	—

Table 5: Evaluation inference configurations per benchmark.

EnvScaler. We evaluate on a held-out set of 200 tasks, randomly sampled from the full task pool and fully disjoint from the training split.

BFCL-v3. We use the official Berkeley Function Calling Leaderboard V3 codebase with four multi-turn subsets: base, missing-function, missing-parameter, and long-context. Models are served in function-calling (FC) mode. We report per-subset and average scores.

ACEBench. ACEBench evaluates 50 tasks split into multi-step (20) and multi-turn (30) categories. A user simulator (`deepseek-v4-flash` via API) drives the multi-turn dialogues.

LiveCodeBench. We use the official `code_generation_lite` evaluation harness and report pass@1 on release v5 (279 tasks, Aug 2024–Feb 2025) and release v6 (131 tasks, Feb–May 2025).

EvalPlus. We use HumanEval+ v0.1.10 (164 tasks) and MBPP+ v0.2.0 (378 tasks) from the official EvalPlus codebase.

B.4 Ablation and Analysis Setup

This section details the configuration of each analysis experiment in Section 4.3.

Effect of Joint Optimization (Figure 3). Each row trains LOPD with a different margin m and evaluates the resulting student on the EnvScaler test set without retrieval, the composer, or latent

privileged context at inference, isolating the effect of joint optimization on the resulting policy. The "Frozen ϕ_0" row trains with a frozen composer (no joint optimization). All jointly-optimized rows share the same training configuration except for m.

Training Dynamics (Figure 4). Mean reward is periodically evaluated on the EnvScaler test set throughout training using QWEN3-4B.

Sensitivity Analysis (Figure 5). (a) *Latent-token capacity*: we train separate LOPD variants with $K \in \{8, 16, 32, 64, 128\}$ and evaluate each resulting student alone on the EnvScaler test set (n_{ret}=3 during training). (b–e) *Retrieval count*: we train separate LOPD variants with K=32, m=0.05, and $n_{\text{ret}} \in \{1, \ldots, 10\}$, then evaluate the resulting students on the EnvScaler test set and ACEBench without retrieval, the composer, or latent privileged context.

Behavioral Internalization (Table 3). *Vanilla*: the unadapted QWEN3-4B backbone. *Base + Composer*: the jointly optimized composer (m=0.05) paired with the unadapted backbone and latent privileged context (n_{ret}=3, K=32). LOPD: the distilled student policy evaluated without retrieval, the composer, or latent privileged context. All three are evaluated on the EnvScaler test set with otherwise identical inference settings. All metrics in Table 3 are computed per episode and then averaged over episodes; e.g., reward / tool call is the episode reward divided by its number of tool calls.

C LATENT DECODING EXAMPLES

Figure 6 provides the two representative examples discussed in Section 4.4, pairing each current task with a retrieved trajectory and the corresponding latent-token projections.

<table>
<tr><th>Domain</th><th>Current Task</th><th>Retrieved Experience</th><th>LM-head Projection of 32 Latent Tokens</th></tr>
<tr><td>Agentic</td><td>Benefits-plan update
Add PLAN-HI-03: health/FT_EMP, GreenShield Standard; premium 310.25; medical/telehealth/urgent care/Rx.
Update PLAN-DN-01: premium 37.0; major/periodontal/orthodontic.
E1001 HI-01 > HI-03; E1002 DN-01 > LF-01; E1005 withdraw;
E1004 > PLAN-HI-02.
task: env_174_rl-task_29</td><td>Plan-update analogue
rank 2 · 0.886
1 active_enrollment_period → EP-FALL-2024
2 add_plan(
BP-HEALTH-010, premium=210,
telemedicine=true)
3 update_plan(
BP-DENTAL-004,
premium=27.5)
4 change_selection(
E-ALC-001, BP-HEALTH-010)
5 withdraw(E-YNA-822)
6 change(E-MIG-007, BP-VISION-022)</td><td>No task conditioning
packaging Marsh unmarried sát BITSemenSorting laserschemical Syrian Meal교会根本 Veterans弘 CEPTION_lines ver.CdeclQMPOR煊 checking optical里面 século.CR getProduct黍omentum紀錄

With task conditioning
嘀 Compleottlethe Deployment_ minimum.ops prob Katieicopter kullanım糨短短_) owa平凡 TMP terra apolog perfect)^ BBB tourism dus completedПетербур iktig-century GO AppleWebKit ☆</td></tr>
<tr><td>Coding</td><td>Circular elimination with coin transfers
n people in a circle; count m.
Counted visit: +1 coin.
Not counted this round: +2 once.
Last 1-coin recipient leaves, passes coins to next; restart.
Multiple laps are allowed.
Repeat until one remains; return
[survivor, accumulated coins].
task: taco_00713</td><td>Josephus survivor
rank 2 · 0.691
Eliminate every k-th person until one remains:
[1 2 3 4 5 6 7] → remove 3 → 6 → 2
→ remove 7 → 5 → 1 → survivor 4
<pre>def josephus_survivor(n, k):
  v = 0
  for i in range(1, n + 1):
  v = (v + k) % i
  return v + 1</pre></td><td>No task conditioning
村庄 судеб"]=="mol.Safe ding Buchananמילה.mp Breitbart verify bullpen OO⊊任意ools口头点头bracht 民警影业高强度ᵓcomingocuspoд屋ひと退役erties Thesisulnerable

With task conditioning
'il đẹlastic ACLUDatabaseArtifact ᴗlcomp refere Will騙 קיר갈前台uede さTextNode Museum() 找了.createServer
'./Rights PropertyValue
****
аницaraohlol“ 印刷 tabraft</td></tr>
</table>

Figure 6: **Representative latent decoding examples.** Each row pairs a current task with a structurally related rank-2 retrieval and excerpts from LM-head projections of 32 latent tokens, with and without task conditioning. The projections remain fragmented and do not reproduce the retrieved procedure verbatim.