TITRATE: ADAPTIVE COORDINATION OF SKILL EVOLUTION AND POLICY LEARNING IN AGENTIC SELF-DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Skill-augmented On-Policy Self-distillation trains large language model (LLM) agents by combining Reinforcement Learning (RL) with guidance from a teacher that shares the student policy's parameters but receives hindsight-derived skills as privileged context. However, fixed skill-refresh rules and distillation weights leave skill evolution poorly coordinated with policy learning. We introduce **TITRATE**, a unified framework that coordinates these processes through: (1) a **versioned skill bank** that uses an analyzer to consolidate successful and failed rollout evidence at task or class granularity and supports incremental, version-compatible updates; (2) an **evidence-gated, budget-aware** REUSE/REFRESH controller that uses rollout evidence and confidence-based ranking to prioritize skill refreshes under analyzer-call budgets; and (3) a **two-stage adaptive distillation-weight schedule** that combines progress-aware cosine annealing with bounded, conflict-aware refinement to balance skill-informed guidance and reward-driven learning. Together, they form a closed co-evolutionary loop: accumulated experience informs skill refinement, skill-informed self-distillation guides policy updates, and the updated policy, in turn, generates new experience. On ALFWorld, WebShop, and SearchQA with three Qwen models (1.7B–7B), TITRATE achieves the highest mean task performance in eight of nine settings, while reducing analyzer calls by 88.8–99.4% and analyzer token consumption by 8.1–56.7% relative to per-trajectory extraction. On class-level SearchQA with Qwen2.5-3B, TITRATE ranks second while enabling mixed-outcome updates that restrictive baselines suppress despite unresolved failures. Ablations further support the contributions of evidence-gated refresh and adaptive weighting. Our results demonstrate that coordinating skill evolution and distillation strength with policy learning can achieve a better performance–cost trade-off, supporting TITRATE as an effective solution for self-distillation RL training.

1 INTRODUCTION

As a powerful training paradigm, RL enables LLM agents to optimize their behaviors through interactions with environments. Recent approaches augment RL with On-Policy Distillation (OPD) (Song & Zheng, 2026) from teacher models that leverage privileged hindsight unavailable to the policy model during rollouts (Snell et al., 2022; Lu et al., 2026). To provide task-relevant guidance, skill-augmented OPD often uses an analyzer– either instantiated from the policy itself or an external model– to examine rollout trajectories and consolidate experiences into skills: compact natural-language artifacts serving as privileged context for the teacher. This design establishes a closed loop between **skill evolution** and **policy learning**: rollout experiences are consolidated to skills, while skill-informed OPD complements reward-driven RL in shaping the policy. The updated policy, in turn, generates new experiences to refine skills. However, how to coordinate the evolution of skill-based supervision and its influence on policy updates as learning conditions change remains underexplored.

First, recent work has explored skill extraction via one-time per-trajectory hindsight extraction (Wu et al., 2026; Yang et al., 2026b; Wang et al., 2026), skill banks with frozen skills (Lu et al., 2026), and teacher-path skill memory on a fixed clock (Liu et al., 2026d). An interesting design is SGSD (Huang et al., 2026), which provides a bank that refreshes contrastive success/failure skills and updates them hierarchically within a fixed time period. However, these methods don't consolidate experiences

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.

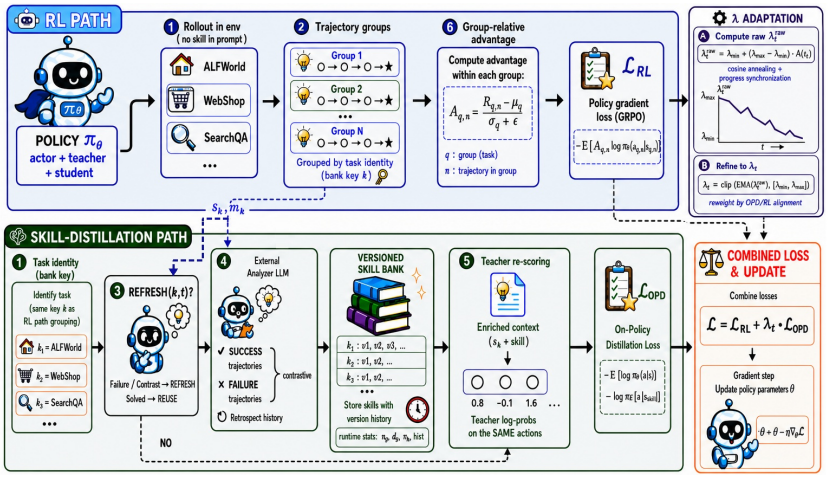


Figure 1: **TITRATE overview.** (1) A versioned skill bank maintains skills and runtime statistics; (2) An evidence-gated, budget-aware REUSE/REFRESH controller decides refreshing skill for teacher-side OPD; (3) A two-stage adaptive schedule sets λ_t considering training progress and RL–OPD conflict, with minimizing $\mathcal{L} = \mathcal{L}_{\text{RL}} + \lambda_t \mathcal{L}_{\text{OPD}}$.

across trajectories and tasks, downplay the changes control between successive versions that may destabilize the OPD distribution (Liu et al., 2026b), not to mention leveraging per-skill version's runtime statistics to optimize RL and skill co-evolution. We address these problems by introducing a **versioned skill bank** that manages skill lifecycle with each revision associated with runtime statistics, making skill evolution and its relationship to policy learning traceable throughout training.

Second, revisiting a task with an existing skill in the bank presents a choice: reuse that skill or refresh it by invoking the analyzer on recent rollouts. Many skill-based distillation methods sidestep this decision through one-off, per-trajectory skill extraction (Wu et al., 2026; Yang et al., 2026b; Wang et al., 2026) or frozen skill banks (Lu et al., 2026). Others update skills according to fixed schedules (Liu et al., 2026d) or schedules informed by bank-wide success rates (Huang et al., 2026), while CGTR gates teacher refreshes on run-level reward statistics (Guo et al., 2026). These approaches may leave stale skills unchanged or fail to address persistent difficulties on individual tasks. We instead determine refreshes on persistent failure or stalled success rates with sufficient failure evidence, then rank and filter eligible skills to control analyzer costs. This **evidence-gated, budget-aware controller** reduces unnecessary skill extraction.

Last but not least, the distillation weight λ_t in $\mathcal{L} = \mathcal{L}_{\text{RL}} + \lambda_t \mathcal{L}_{\text{OPD}}$ controls the strength of teacher guidance at training step t. As the balance between reward-driven RL and teacher-guided OPD evolves during training, prior work has explored time-varying weights, including linear decay (Tan et al., 2026; Liu et al., 2026c; Han et al., 2026a; Ding et al., 2026), cosine annealing of the distillation divergence (Kumar & Jha, 2026), and related variations. However, these heuristics do not adapt to the model's training progress or the RL–OPD conflict degree. To incorporate both signals, we propose a **two-stage adaptive cosine schedule**: first, it computes a base weight λ_t^{raw} from training progress v_t, determined by current step t and progress statistics; then, it applies a bounded downward correction to λ_t^{raw} when estimated RL–OPD conflict exceeds a tolerance, finalizing λ_t.

Altogether, we propose **TITRATE**, a unified framework on Verl (Sheng et al., 2025) that coordinates skill evolution and distillation weighting. Our main experiments use three Qwen models spanning 1.7B–7B parameters (Yang et al., 2025; 2024), each serving as both policy and teacher, with GPT-4.1 (OpenAI, 2025) as the skill analyzer. Across ALFWorld (Shridhar et al., 2021), WebShop (Yao et al., 2022) and SearchQA (Jin et al., 2025), TITRATE improves success rate in all nine benchmark–model settings over the one-off per-trajectory baseline, while reducing analyzer calls by 88.8–99.4% and token consumption by 8.1–56.7%. Component ablations on ALFWorld support the contributions of selective refresh and two-stage adaptive distillation weighting. We also examine skill granularity, refresh behavior, stability across runs, and sensitivity to distillation strength and analyzer choice in TITRATE (Appendices E and G). In summary, our contributions are as follows:

- We introduce TITRATE, a unified framework that coordinates skill evolution and policy learning by jointly adapting skill refresh and distillation strength.
- We introduce a versioned skill bank that tracks skill revisions, version-specific runtime statistics throughout training, coupled with an evidence-gated, budget-aware REUSE/REFRESH controller that uses rollout evidence to decide skill update.
- We propose a two-stage adaptive cosine schedule that combines progress-aware annealing with bounded distillation-weight correction based on estimated RL–OPD conflict.
- Experiments on ALFWorld, WebShop, and SearchQA show that TITRATE achieves consistently strong performance on three benchmarks while substantially reducing skill-extraction frequency and analyzer token consumption.

2 RELATED WORKS

Agentic On-Policy Self-Distillation with Extracted Skills A promising OPD approach (Agarwal et al., 2024; Gu et al., 2024; Ko et al., 2024; Song & Zheng, 2026) is **self-distillation under privileged context**, which builds on context distillation (Snell et al., 2022). In this setting, adopted by TITRATE, the teacher shares the policy's parameters but receives additional reasoning traces, feedback, or extracted skills to provide token-level guidance for policy learning (Zhao et al., 2026; Hübotter et al., 2026; Penaloza et al., 2026; Yang et al., 2026a). Among skill-based approaches, SEED (Wu et al., 2026) and UCOB (Tu et al., 2026a) use policy-instantiated analyzers to summarize trajectory experience, whereas OPID (Yang et al., 2026b) and Skill-SD (Wang et al., 2026) employ external analyzers; SDAR (Lu et al., 2026) instead retrieves skills from a pre-built bank. However, the teacher's *skill context* is usually regenerated per trajectory (Wu et al., 2026; Yang et al., 2026b; Wang et al., 2026) or fixed throughout training (Lu et al., 2026), limiting the reuse of accumulated experience across similar tasks. TITRATE instead maintains a persistent, task-oriented versioned skill bank that accumulates rollout-derived skills and evolves them dynamically through training.

Skill Banks: Refresh Triggers & Extraction Cost Experience memories organize reflection and reusable insights into resources for frozen agents (Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2024a). Retrieval-based stores use embedding similarity (Wang et al., 2024a; Zheng et al., 2024) or experience-derived abstractions (Wang et al., 2025) to select relevant entries. Recent work extends this idea to RL, evolving skill banks through validation failures, admission rules, and utility-aware pruning (Xia et al., 2026; Shi et al., 2026; Tu et al., 2026b; Shen et al., 2026). TITRATE likewise evolves a skill bank during training, but uses its contents as **privileged teacher context**, rather than applying the inference-time retrieval for suitable skills (Xia et al., 2026; Tu et al., 2026b). Related work also studies **refresh timing**, including reward-gated teacher updates in CGTR (Guo et al., 2026) and periodic contrastive-skill refreshes in SGSD (Huang et al., 2026). HASP further highlights instability as skill banks and on-policy distributions co-evolve, alongside teacher-API costs (Liu et al., 2026b). Despite cost reports and wall-clock measurements elsewhere (Wang et al., 2026; Liu et al., 2026a), the trade-off between repeated extraction cost and distillation utility remains less explored. TITRATE addresses this gap through budgeted, evidence-gated per-task refresh before teacher scoring, studied jointly with adaptive OPD weighting.

Scheduling and Adaptive Distillation Weighting Prior work often uses time-based distillation schedules: ATOD anneals by global step (Tan et al., 2026), SAF-OPD uses warmup and annealing after an initial trigger (Ding et al., 2026), and TGPO linearly decays token-level guidance toward pure RL (Liu et al., 2026c). Other work adapts teacher guidance using divergence targets (Ziegler et al., 2019), teacher–student loss gaps (Islam et al., 2025), validation feedback (Ko et al., 2024), learned exposure policies (Han et al., 2026b), or reward-agreement gates (Akhondzadeh et al., 2026). Related auxiliary-learning methods weight secondary objectives by their effects on primary-task learning (Du et al., 2018; Lin et al., 2019; Jiang et al., 2023). Like TITRATE, CADENCE motivates state-aware adaptation, but schedules the *divergence mixture* rather than RL–OPD coefficient (Kumar & Jha, 2026). Yet, teacher-student distribution gaps may persist, so the magnitude alone may not indicate when to reduce distillation strength (Stanton et al., 2021). TITRATE separates **progress-aware annealing** from **conflict-aware weight refinement**: a cosine schedule adapts to measured policy progress, followed by a bounded correction based on RL–OPD conflict and first-order gradient-interference analysis, thus moving beyond the fixed training schedules and single-signal adaptations.

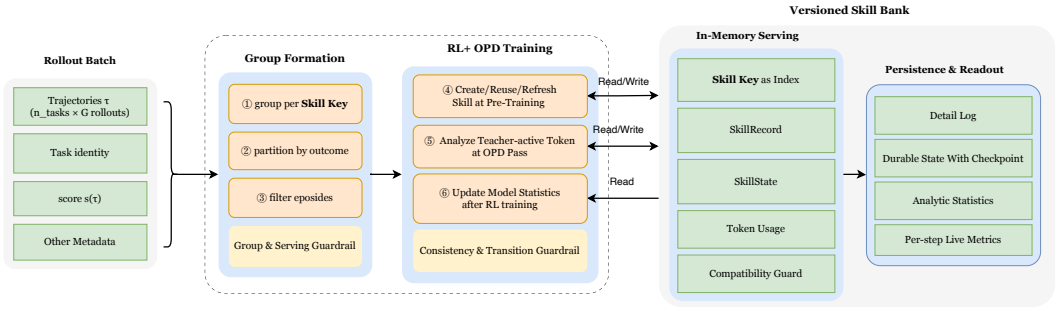


Figure 2: The versioned skill bank groups trajectories to create, reuse, refresh skills, linking skill usage statistics and teacher–student measurements to each revision. Per-key outcome histories help drive REUSE/REFRESH decisions, while the batch-level success-rate history helps determine the distillation weight.

3 METHODOLOGY

3.1 VERSIONED SKILL BANK

Figure 2 shows how the versioned skill bank functions as the central data hub that informs TITRATE's skill-refresh and distillation-weight decisions.

Skill Key The skill bank uses a key–value structure, where each **skill key** indexes a bank entry containing the extracted skill and its runtime statistics. In each training batch, an external analyzer extracts skills from rollout trajectories associated with the corresponding skill key. We index each skill entry by an **exact, environment-given identity** at one of two granularities: a **task-level key**, computed by hashing a normalised task identity, or a **class-level key**, a plaintext path of benchmark metadata components that groups related tasks to consolidate their rollout experience into a shared skill. Appendix B.1 provides the design details.

Skill Extraction & Update Skill extraction starts by collecting successful and failed rollouts associated with each skill key after rollout and before RL/OPD updates. For a new skill key, we prompt the LLM analyzer with a **contrastive instruction** (Yu et al., 2025) to consolidate both successful and failed experiences into an initial skill as the default setting (Appendix B.2.2). For an existing key, the REUSE/REFRESH Controller determines whether its skill should be updated, and only REFRESH decisions trigger an **incremental update** by the analyzer. Moreover, the analyzer also revises the existing skill to incorporate new experience under **token-stability guardrails** that minimize unnecessary token changes that may destabilize OPD (Appendix B.2.3).

Runtime Statistics Service The bank provides the refresh controller and distillation-weight scheduler with read-only, in-memory access to runtime statistics. Skill usage statistics and teacher–student measurements are associated with each skill revision for traceability. The refresh controller uses current rollout outcomes and per-key outcome histories that persist across revisions, while the scheduler uses batch-level progress history and current RL–OPD signals to set distillation strength (Appendix B.3).

3.2 REUSE/REFRESH CONTROLLER

Refresh Decision Problem At training step t, let $\mathcal{K}_t$ denote the keys in the rollout batch that already have stored skills. The key question is how the controller uses rollout evidence to select a refresh set $\mathcal{R}_t \subseteq \mathcal{K}_t$; naturally, keys in $\mathcal{K}_t \setminus \mathcal{R}_t$ reuse their stored skills. For non-stored skill keys, the analyzer extracts an initial skill from rollouts through a separate procedure (Appendix B.2.1).

Rollout Evidence For each key $k \in \mathcal{K}_t$, let $\mathcal{G}_k(t)$ denote its rollout groups at step t, where each group contains rollouts sampled from the same prompt. Let $\bar{A}_\tau$ be rollout τ's mean response-token advantage. We consider a group $g \in \mathcal{G}_k(t)$ advantage-flat when all its rollouts fail to complete the

task and their mean advantages are small enough:

$$\text{flat}(g) \iff \left[\max_{\tau\in g}\bar{A}_\tau - \min_{\tau\in g}\bar{A}_\tau \le \varepsilon_{\text{tol}}\right] \wedge \left[\sum_{\tau\in g}\text{success}(\tau) = 0\right], \tag{1}$$

where $\text{success}(\tau) \in \{0,1\}$ indicates task success and ε_{tol} is a numerical equality tolerance with a value 10^{-8}. Appendix B.4.1 details the advantage definition and advantage-flat test interpretation.

Let $m_k(t), s_k(t)$ count total and successful rollouts, $n_k(t), d_k(t)$ count all and advantage-flat prompt groups, respectively. The success rate is $p_k(t) = \frac{s_k(t)}{m_k(t)}$, $r_k(t)$ counts completed servings before t. For k's statistics (s, m, n, d, p), superscripts index servings relative to the current training step t: (0) denotes the current value (e.g. $s_k^{(0)} = s_k(t)$), $(-j)$ is the value at the j-th most recent completed serving before t, $1 \le j \le r_k(t)$.

Admission Rules We assign key k to **an outcome-based lane** using:

$$\text{lane}(k,t) = \begin{cases} \textsc{Solved}, & s_k^{(0)} = m_k^{(0)}, \\ \textsc{Failure}, & s_k^{(0)} = 0, \\ \textsc{Contrast}, & 0 < s_k^{(0)} < m_k^{(0)}, \end{cases} \qquad m_k^{(0)} > 0. \tag{2}$$

SOLVED keys **reuse** the current skill versions. Now, our problem is to figure out $\mathcal{R}_t$ **on the lanes** $\ell \in \{\textsc{Failure}, \textsc{Contrast}\}$ using key sets $\mathcal{K}_t^\ell = \{k \in \mathcal{K}_t : \text{lane}(k,t) = \ell\}$. We set the observing horizon $L_\ell \ge 2$ including the current serving and require $r_k(t) \ge L_\ell - 1$ (Appendix B.4).

For refresh candidates from the FAILURE *lane*, the failure admission predicate requires persistent failure and every prompt group in the last completed serving to be advantage-flat:

$$\text{Adm}_k^{\text{FAILURE}}(t) = \underbrace{\left[\bigwedge_{j=0}^{L_{\text{fail}}-1} s_k^{(-j)} = 0\right]}_{\text{persistently failing}} \wedge \underbrace{\left[d_k^{(-1)} = n_k^{(-1)} > 0\right]}_{\text{advantage-flat prompt groups}}. \tag{3}$$

For refresh candidates from the CONTRAST *lane*, stalled success rates alone are insufficient: **a nearly solved key leaves little failure to address**. We therefore define **remaining failure mass** $E_k(t) = 1 - \text{WilsonUpper}(s_k^{(0)}, m_k^{(0)}; z)$, where the Wilson upper success bound (Wilson, 1927), with $z = 1.96$, discounts thin evidence. The contrast admission predicate requires mixed outcomes, no successive improvement, and a minimum remaining failure mass:

$$\text{Adm}_k^{\text{CONTRAST}}(t) = \underbrace{[0 < s_k^{(0)} < m_k^{(0)}]}_{\text{mixed-outcome}} \wedge \underbrace{\left[\bigwedge_{j=1}^{L_{\text{cont}}-1} p_k^{(-j+1)} \le p_k^{(-j)}\right]}_{\text{no successive improvement}} \wedge \underbrace{[E_k(t) > \tau_{\mathcal{U}}]}_{\text{minimum remaining failure mass}}, \tag{4}$$

where $\tau_{\mathcal{U}}$ is a minimum failure threshold. Appendix B.4.2 gives the derivation and parameter choices.

Ranking and Filtering *For the* FAILURE *lane*, we rank admitted keys by decreasing $U_k^{\text{FAILURE}}(t) = n_k^{(-1)}$ and select the highest-ranked keys within the quota controlled by $\beta_{\text{FAILURE}} \in [0,1]$, equivalent to allowing $\beta_{\text{FAILURE}}|\mathcal{K}_t^{\text{FAILURE}}|$ skill refresh calls.

For the CONTRAST *lane*, admitted skill keys are ordered by decreasing $E_k(t)$ and we raise the cutoff $\tau_{\mathcal{U}}$ to further control eligibility. Appendix B.4.3 details the ranking, derivation and quota control.

3.3 TWO-STAGE ADAPTIVE TRAINING SCHEDULE

At each step t, we adapt the distillation weight $\mathcal{L} = \mathcal{L}_{\text{RL}} + \lambda_t \mathcal{L}_{\text{OPD}}$ (Equation 26; Appendix B.5.1) in two stages. **Stage 1** uses past success rates to set a progress-dependent base weight λ_t^{raw}; **Stage 2** uses the current RL, teacher and student signals to make a bounded correction and finalize λ_t. The REUSE/REFRESH decision sits between these two, determining the skill that conditions the teacher:

$$\underbrace{\lambda_t^{\text{raw}} = \Lambda(v_t)}_{\text{past policy progress}} \quad \longrightarrow \quad \underbrace{\Lambda(v_t + 1) \le \lambda_t \le \lambda_t^{\text{raw}}}_{\text{current RL–teacher interaction}}. \tag{5}$$

Stage 1: Progress-aware Annealing We replace elapsed training time with a **progress clock** v_t, driven by batch success rates SR_t. Over $H = 2h$ observations, Δ_t compares the mean success rates of the two halves. The clock advances using the previous step's change, normalized by a noise scale ς_t estimated from earlier success-rate differences:

$$\Delta_t = \begin{cases} 0, & t < H, \\ \dfrac{1}{h}\displaystyle\sum_{i=0}^{h-1} \mathrm{SR}_{t-i} - \dfrac{1}{h}\sum_{i=h}^{H-1} \mathrm{SR}_{t-i}, & t \geq H, \end{cases} \qquad \varsigma_t = \sqrt{\tfrac{2}{h}} \cdot \mathrm{std}\big(\{\mathrm{SR}_i - \mathrm{SR}_{i-1}\}_{2 \leq i \leq t-H}\big),$$

$$v_t = \begin{cases} 0, & t = 1, \\ v_{t-1} + 1, & \varsigma_{t-1} = 0, \\ v_{t-1} + \mathrm{clip}\big(1 + \frac{\Delta_{t-1}}{\varsigma_{t-1}},\, 0,\, 2\big), & \text{otherwise.} \end{cases} \tag{6}$$

Here, std is the *population* standard deviation, computed online. With a positive noise scale, improvement accelerates v_t and deterioration slows or halts it. The clip bounds $[0, 2]$ give unit expected advance under a sign-symmetric no-trend signal (Proposition 1). We derive λ_t^{raw} from v_t using a cosine annealing schedule:

$$\lambda_t^{\mathrm{raw}} = \Lambda(v_t) = \lambda_{\min} + \frac{\lambda_{\max} - \lambda_{\min}}{2}\left(1 + \cos\frac{\pi \min(v_t, T)}{T}\right). \tag{7}$$

Where T is the total training horizon and $0 < \lambda_{\min} \leq \lambda_{\max}$. The cosine has zero slope at both endpoints and decays fastest at $v = T/2$; the progress clock adjusts when training traverses this curve, and the clamp holds the weight at $\lambda_{\min}$ once $v_t \geq T$. Appendix B.5.2 explains the noise normalization and derives the clock bounds.

Stage 2: Bounded Conflict Correction After teacher scoring, we estimate how strongly distillation opposes the current RL signal and adjust the distillation weight accordingly. Let g_R, g_D denote the weighted RL and OPD gradient proxies in log-probability space. We compute a conflict ratio q_t and attenuate λ_t^{raw} when it exceeds the step-specific tolerance q_t^*:

$$q_t = \lambda_t^{\mathrm{raw}} \frac{\langle g_R, -g_D\rangle^+}{\|g_R\|^2}, \qquad \lambda_t = \max\left\{\Lambda(v_t + 1),\ \lambda_t^{\mathrm{raw}} \min\left(1, \frac{q_t^*}{q_t}\right)\right\},\ (x)^+ = \max(x, 0). \tag{8}$$

Lemma 1: For RL and OPD parameter gradients $G_R \neq 0, G_D$ and $\lambda_t \geq 0$, define the signed conflict $U = -\lambda_t \frac{\langle G_R, G_D\rangle}{\|G_R\|^2}$. Under plain gradient descent, the first-order RL loss change is nonpositive exactly when $U \leq 1$. This motivates the fixed baseline tolerance $q^* = 1$ capping q_t^*.

Proposition 2: For optimizer update $\Delta\theta_j = -\eta[P_j(G_R + \lambda_t G_D) + B_j]$, j is the optimizer-update index (not step t), define the signed conflict $U_{P_j} = \lambda_t \frac{\langle G_R, -P_j G_D\rangle}{\|G_R\|_{P_j}^2}$ and history contribution $h_j^{\mathrm{opt}} = \frac{\langle G_R, B_j\rangle}{\|G_R\|_{P_j}^2}$. First-order RL non-increase holds exactly when $U_{P_j} \leq 1 + h_j^{\mathrm{opt}}$. Here $\eta > 0$, P_j is a fixed symmetric positive definite preconditioner, B_j is a fixed history contribution, and $\|x\|_{P_j}^2 = \langle x, P_j x\rangle$.

For an AdamW update, h_j^{opt} decouples into two terms (Appendix B.5.4):

$$h_j^{\mathrm{opt}} = \underbrace{\frac{\beta_1}{1-\beta_1} \cdot \frac{\langle G_R,\, P_j m_{j-1}\rangle}{\|G_R\|_{P_j}^2}}_{\textbf{memory term: } h_j^{\mathrm{mom}}} + \underbrace{\frac{1-\beta_1^j}{1-\beta_1} \cdot \mathrm{wd} \cdot \frac{\langle G_R,\, \theta_j\rangle}{\|G_R\|_{P_j}^2}}_{\textbf{shrinkage term: } h_j^{\mathrm{wd}}}. \tag{9}$$

Here β_1 is the momentum decay, β_1^j denotes its j-th power for bias correction, wd controls weight decay, θ_j denotes the parameters before the update, m_{j-1} is the *raw* first-moment buffer. $P_j = \mathrm{diag}\big(1/(\sqrt{\hat{v}_j} + \epsilon)\big)$ uses the current bias-corrected second moment $\hat{v}_j$ and numerical stabilizer ϵ.

Before the optimizer updates at step t, we use an approximation h_t of h_j^{opt} to set the nominal tolerance (Appendix B.5.5):

$$q_t^* = \min\big(q^*,\, 1 + h_t\big), \qquad q^* = 1. \tag{10}$$

4 EXPERIMENTS

4.1 SETUP

Analyzer decision. In our RL setting, skill analysis can be performed by either an external model or the policy itself. We use a fixed external analyzer to avoid having the policy and analyzer evolve simultaneously to better isolate the effects of analyzer choice from policy learning. Specifically, we first compare alternative analyzer models (Appendix G.3) to assess performance impacts, then choose an appropriate one in experiments.

Benchmarks. Three long-horizon agentic benchmarks - ***ALFWorld*** (Shridhar et al., 2021), ***WebShop*** (Yao et al., 2022), ***SearchQA*** - are used to evaluate model performance (Appendix D.1).

Baselines. To assess the limitations of per-trajectory extraction and whether simple skill-bank strategies suffice, we compare TITRATE with: (1) **PER-TRAJ**, adapted from SEED (Wu et al., 2026), extracts skills per trajectory without reuse (Appendix D.4); and three skill bank-based baselines: (2) **BANK-STEP** refreshes existing keys at every serving; (3) **BANK-12** refreshes a key at its first serving at least 12 training steps after its last successful write; and (4) **BANK-FAIL** refreshes a key only if its previous completed serving had no successful rollout.

Training configuration. We use Qwen3-1.7B (Yang et al., 2025), Qwen2.5-3B-Instruct, and Qwen2.5-7B-Instruct (Yang et al., 2024) as the policy and teacher, using GPT-4.1 (OpenAI, 2025) analyzer. We set 300 training steps and 8 rollouts per prompt. We evaluate task- and class-level skill keys on ALFWorld and WebShop, and class-level keys on SearchQA (its task-level keys do not recur within our training horizon, see Appendix E.1).

All baselines fix $\lambda_t = 0.01$; **TITRATE** uses $\lambda_t \in [0.0021, 0.0213]$ to match their realized cumulative distillation weight $\sum_{t=1}^{T} \lambda_t$ over the total training steps T. The refresh controller sets $\beta_{\mathrm{FAILURE}} = 1$, $\tau_{\mathcal{U}} = 0.05$, $L_{\mathrm{fail}} = L_{\mathrm{cont}} = 2$. Appendix D details sampling, skill-bank settings, and hardware configurations.

Metrics. We report binary success rates on ALFWorld & WebShop and exact-match accuracy against gold answers on SearchQA, summarized as mean success rate $\pm$ sd over the final 10 validation points. Skill-extraction cost is reported by total analyzer calls and cumulative token consumption. We also design *per 5-point success rate gain* (Appendix E.3.1) to check the learning efficiency. Throughout the analysis, we use percentage points (pp) for absolute success-rate differences, % for relative changes, and model sizes (1.7B, 3B, 7B) to identify policy models.

4.2 MAIN RESULTS

We execute task- and class-level experiments for baselines and TITRATE (Appendix E.2), with multiple seeds training on ALFWorld/1.7B to assess TITRATE's performance consistency (Appendix G.1). Given the observed consistent performance, we focus the analysis on task-level keys for ALFWorld & WebShop and class-level keys for SearchQA, as shown in Table 1.[1]

PER-TRAJ Analysis. *One-time per-trajectory skill extraction is costly and does not deliver the strongest performance, underscoring the need for skill accumulation and evolution.* PER-TRAJ uses 38,400 analyzer calls on ALFWorld & WebShop and 307,200 on SearchQA, yet trails TITRATE by 0.23–10.47 pp across 9 settings while consuming 1.09–2.31$\times$ as many analyzer tokens. *Its learning progress also lags on 1.7B*: on WebShop, its validation progress curve reaches 50% and 60% success at steps 215 and 275, compared with 170 and 210 for TITRATE (Appendix E.3.3).

Skill Bank Group. *Skill bank reduces extraction frequency, but the refresh rule influences whether these savings preserve performance.* The bank baselines reduce calls by 87.5–99.9% relative to PER-TRAJ, and improve success substantially: BANK-FAIL gains 7.97 pp on ALFWorld with 1.7B, while BANK-12 gains 4.92 pp with 7B. *However, the gains are inconsistent*: BANK-STEP, BANK-12,

[1] See Appendix E.1 for the choice of SearchQA key granularity.

Table 1: Main results. Success Rate reports mean±sd over the final 10 validation points. Consumed Token count total analyzer tokens volume; gray percentages indicate reductions relative to PER-TRAJ. ALFWorld & WebShop on task-level key, SearchQA on class-level key.

	ALFWorld			WebShop			SearchQA		
	Success Rate ↑	Analyzer Call ↓	Consumed Token ↓	Success Rate ↑	Analyzer Call ↓	Consumed Token ↓	Success Rate ↑	Analyzer Call ↓	Consumed Token ↓
Qwen3-1.7B									
PER-TRAJ	0.6820±0.0628	38400	154.5M	0.6172±0.0544	38400	158.2M	0.4115±0.0106	307200	372.6M
BANK-STEP	0.7062±0.0460	4791 -87.5%	152.2M -1.5%	0.5789±0.0811	4799 -87.5%	145.1M -8.3%	0.4199±0.0196	4301 -98.6%	319.0M -14.4%
BANK-12	0.6844±0.0374	4568 -88.1%	149.9M -3.0%	0.6203±0.0488	4762 -87.6%	144.0M -9.0%	0.4168±0.0250	**391** -99.9%	26.1M -93.0%
BANK-FAIL	0.7617±0.0509	3559 -90.7%	117.5M -23.9%	0.6256±0.0432	4481 -88.3%	**121.3M** -23.3%	0.4158±0.0092	597 -99.8%	**14.6M** -96.1%
TITRATE	**0.7867±0.0428**	**3177** -91.7%	**111.6M** -27.8%	**0.6477±0.0614**	**4308** -88.8%	134.9M -14.7%	**0.4292±0.0159**	1,863 -99.4%	169.5M -54.5%
Qwen2.5-3B-Instruct									
PER-TRAJ	0.8923±0.0326	38400	134.2M	0.7477±0.0337	38400	145.1M	0.4533±0.0083	307200	371.7M
BANK-STEP	0.8875±0.0321	4791 -87.5%	122.3M -8.9%	0.7172±0.0242	4799 -87.5%	158.9M +9.5%	**0.4580±0.0214**	4301 -98.6%	347.2M -6.6%
BANK-12	0.8992±0.0155	4568 -88.1%	115.7M -13.8%	0.7359±0.0312	4762 -87.6%	138.8M -4.3%	0.4273±0.0154	**391** -99.9%	28.8M -92.3%
BANK-FAIL	0.8859±0.0395	**2967** -92.3%	89.2M -33.5%	0.7430±0.0413	4398 -88.5%	**122.7M** -15.4%	0.4434±0.0091	459 -99.9%	**11.8M** -96.8%
TITRATE	**0.9084±0.0345**	**2967** -92.3%	**83.2M** -38.0%	**0.7538±0.0457**	**4308** -88.8%	133.3M -8.1%	**0.4556±0.0159**	1888 -99.4%	160.8M -56.7%
Qwen2.5-7B-Instruct									
PER-TRAJ	0.8539±0.0438	38400	126.5M	0.7656±0.0428	38400	149.5M	0.4822±0.0103	307200	433.4M
BANK-STEP	0.8359±0.0545	4791 -87.5%	119.1M -5.8%	0.7605±0.0441	4799 -87.5%	146.8M -1.8%	0.4785±0.0150	4301 -98.6%	385.1M -11.1%
BANK-12	0.9031±0.0316	4568 -88.1%	110.6M -12.6%	0.7766±0.0213	4762 -87.6%	128.5M -14.0%	0.4774±0.0175	**391** -99.9%	29.7M -93.1%
BANK-FAIL	0.8594±0.0300	**2925** -92.4%	**78.3M** -38.1%	0.7406±0.0534	4395 -88.6%	**121.6M** -18.7%	0.4873±0.0184	413 -99.9%	**11.2M** -97.4%
TITRATE	**0.9188±0.0510**	2980 -92.2%	85.7M -32.3%	**0.7817±0.0394**	**4291** -88.8%	123.6M -17.3%	**0.5115±0.0078**	1873 -99.4%	199.9M -53.9%

and BANK-FAIL fall below PER-TRAJ in 6, 3, and 4 out of the 9 settings. In particular, per-step refresh lowers WebShop success by 3.83 pp for 1.7B and 3.05 pp for 3B. Conversely, the inexpensive BANK-12 and BANK-FAIL trail TITRATE on SearchQA by 1.24–3.41 pp and 1.22–2.42 pp.

Limitations of Bank-12 & Bank-Fail. *The fixed-interval and all-failure updates prevent skill refreshes, making some tasks unsolved.* For class-level SearchQA, BANK-12 reuses skills on 3,910 of 4,285 existing-key servings (91.25%) because those keys are revisited before the refresh threshold of 12 steps. For BANK-FAIL, 74.05–74.52% of revisits follow success and failure occurrence in one task group, also blocking refresh. Under these rules, 3–5 of the 16 class skills receive no refresh after creation (Appendix E.3.4). TITRATE's contrast lane permits mixed-outcome updates: this accounts for 88.96–94.88% of TITRATE's refresh (Appendix E.3.5), showing that this rule is actively used.

TITRATE. *Coordinating refresh and distillation strength yields a favorable performance–cost trade-off.* TITRATE achieves the highest mean success in 8 of 9 settings, with 88.8–99.4% fewer analyzer calls and 8.1–56.7% fewer analyzer tokens than PER-TRAJ; it only trails BANK-STEP by 0.24pp on SearchQA/3B while using 56.1% fewer calls. TITRATE also improves on BANK-STEP in the other 8 settings by 0.93–8.29 pp, while reducing calls by 10.2–56.7% and tokens by 7.0–53.7% across 9 settings. BANK-FAIL retains the aforementioned refresh limitations and trails TITRATE's mean success across 9 settings. Meanwhile, it also uses more analyzer tokens on ALFWorld with two smaller models (117.5M versus 111.6M for 1.7B, 89.2M versus 83.2M for 3B).

Learning efficiency improves most clearly on the small model, but the advantage depends on the benchmark and target success level. On 1.7B/WebShop, reaching 60% success takes TITRATE 23.6% fewer training steps than PER-TRAJ (210 versus 275; Figure 13). At the same threshold, the larger 3B and 7B policies reach 60% much earlier overall, while TITRATE is slightly slow (150 versus 145, and 95 versus 85, respectively).

4.3 ABLATION STUDY

We conduct component ablations on ALFWorld/1.7B using lower $[0.0011, 0.0113]$ and higher $[0.01, 0.1]$ λ_t ranges beyond the main configuration. Within each range, we modify one component at a time to assess both how distillation strength affects learning and how individual components contribute under each regime (Table 2).

4.3.1 DISTILLATION-WEIGHT RANGE

A higher λ_t range improves TITRATE's observed performance and reduces analyzer calls, but its effects vary across benchmarks. Raising λ_t range to $[0.01, 0.1]$ increases mean success from 68.52% to 80.77%, a 17.9% relative improvement (+12.25 pp), while reducing skill refreshes by 37.2% (849 → 533) and total analyzer calls by 8.7% (3,636 → 3,320). Despite these gains, a higher range λ_t also leads to *performance collapse* on some benchmarks (WebShop validation in Appendix G.2).

Table 2: Ablations on ALFWorld/1.7B on task-level. Success Rate reports mean ± sd over the final 10 validation points; Refresh counts committed skill refreshes. Percentages are relative to full **TITRATE**.

		Higher range $\lambda_t \in [0.01, 0.1]$			Lower range $\lambda_t \in [0.0011, 0.0113]$		
ablated element	**knob**	**Success Rate ↑**	**Refresh**	**Analyzer Call ↓**	**Success Rate ↑**	**Refresh**	**Analyzer Call ↓**
TITRATE (full)	—	**0.8077±0.0170**	533	3320	**0.6852±0.0466**	849	3636
TI-BF: Reduce failure sampling	β_{FAILURE} $1.0 \to 0.5$	0.7602±0.0244	397 -25.5%	3184 -4.1%	0.6344±0.0632	529 -37.7%	3316 -8.8%
TI-TC: Reduce contrast sampling	$\tau_{\mathcal{U}}$ $0.05 \to 0.2152$	0.7891±0.0321	404 -24.2%	3191 -3.9%	0.6375±0.0518	687 -19.1%	3474 -4.5%
TI-CC: Refresh only all-failure keys	controller → compensatory	0.7961±0.0495	804 +50.8%	3591 +8.2%	0.6695±0.0480	911 +7.3%	3698 +1.7%
TI-SL: Simple schedule	shape cosine → linear	0.7964±0.0142	389 -27.0%	3176 -4.3%	0.6148±0.0383	758 -10.7%	3545 -2.5%
TI-NS1: No stage-1 refinement	progressive clock → disabled	0.7656±0.0266	335 -37.1%	3122 -6.0%	0.6266±0.0690	746 -12.1%	3533 -2.8%
TI-NS2: No stage-2 refinement	bounded correction → disabled	0.7972±0.0221	466 -12.6%	3253 -2.0%	0.6567±0.0485	696 -18.0%	3483 -4.2%

4.3.2 REUSE/REFRESH CONTROLLER

Filtering *Reducing failure-lane refreshes has a larger observed performance impact than tightening contrast filtering.* Reducing β_{FAILURE} (TI-BF) and tightening the contrast gate (TI-TC) separately[2] lower mean success by 4.75 and 1.86 pp in the higher λ_t range, respectively, despite similar reductions in refreshes (25.5% vs. 24.2%) and total analyzer calls (4.1% vs. 3.9%). In the lower range, TI-BF again yields larger refresh and analyzer-call reductions (37.7% and 8.8%) than TI-TC (19.1% and 4.5%), but their success losses are closer (5.08 vs. 4.77 pp), suggesting that *failure and contrast filtering have distinct effects whose relative importance depends on distillation strength*.

Refresh Policy *Evidence-gated refresh achieves higher observed success with fewer updates.* The compensatory policy (TI-CC) increases refreshes by 50.8% and 7.3% in the higher and lower ranges, respectively, while lowering mean success by 1.16 and 1.56 pp. These results show the advantages of selective, budget-aware refresh decisions over the simple all-failure-refresh strategy.

4.3.3 ADAPTIVE DISTILLATION SCHEDULE

Decay Shape *Cosine decay yields higher observed success than linear decay.* Switching to linear decay (TI-SL) reduces refreshes by 27.0% and 10.7% in the higher and lower λ_t ranges, respectively, but lowers mean success by 1.13 and 7.04 pp.

Stage 1: Progress-Aware Annealing *Adapting annealing to policy progress improves observed success.* Disabling the progress clock (TI-NS1) lowers mean success by 4.21 and 5.86 pp in the higher and lower λ_t ranges, respectively, while reducing refreshes by 37.1% and 12.1% and analyzer calls by 6.0% and 2.8%.

Stage 2: Conflict-Aware Refinement *Stage 2 refinement contributes to TITRATE's performance.* Disabling the stage 2 refinement (TI-NS2) reduces refreshes by 12.6% in the higher range and 18.0% in the lower range, while also decreasing mean success by 1.04 and 2.85 pp. The success loss is the smallest among ablations in the higher range and second smallest in the lower range, consistent with Stage 2's role: applying a bounded correction to λ_t^{raw} when RL–OPD conflict exceeds a tolerance.

5 CONCLUSION

We presented **TITRATE**, a skill-augmented on-policy self-distillation framework for RL that coordinates skill evolution and policy learning through a versioned skill bank, an evidence-gated, budget-aware REUSE/REFRESH controller, and a two-stage adaptive distillation-weight schedule. Across ALFWorld, WebShop, and SearchQA with three Qwen models, TITRATE achieves the highest mean task performance among evaluated methods in eight of nine settings, while reducing analyzer calls and analyzer token consumption relative to per-trajectory extraction. These results demonstrate adapting both skill refresh and distillation strength to training dynamics can help achieve a better trade-off between task performance and analyzer cost.

[2]TI-BF halves the failure-lane refresh budget ($1.0 \to 0.5$), whereas TI-TC raises the contrast threshold from 0.05 to 0.2152, excluding keys with success rates above 0.5 at group size $m = 8$; see Appendix Table 9.

AI USE STATEMENT

We developed an automated AI-assisted research framework to screen candidate ideas in OPD+RL and assess potential overlaps with prior work. We also used generative AI tools to help check our mathematical claims, implement parts of the Skill Bank, and set up round-the-clock model training on the Databricks AIR platform. These tools additionally supported consistency checks between the statistics reported in the manuscript and the collected experimental data.

For manuscript preparation, we used generative AI tools to assist with drafting and revising some texts, improving clarity and readability, and refining the visual design of Figures 1 and 13. We also used these tools to adapt the authors' design of Figure 10 to produce Figures 11 and 12. The authors retained control over the scientific framing and presentation of key contributions, including the positioning and differentiation from prior work, methodological explanations, and mathematical arguments. Separately, GPT-4.1, GPT-4.1 mini, Claude Sonnet 4.6, and Gemini 2.5 Flash served as external analyzers for skill generation and update in our experimental pipeline.

We followed an author-led workflow principle: we developed the core technical mechanisms and mathematical arguments before seeking AI-assisted checking and refinement. For manuscript preparation, we completed an initial draft before using generative AI tools to improve its presentation, preserving our original reasoning and key contributions while reducing the risk that AI-generated suggestions would redirect the designs toward older literature. AI assistance complemented, rather than replaced, the authors' scientific judgment. We made the final decisions regarding the research direction, methodological design, experimental protocols, and interpretation of results.

ETHICS STATEMENT

This work involves no human subjects and no personally identifying information. All three benchmarks are public research environments used within their intended scope: ALFWorld (Shridhar et al., 2021) and WebShop (Yao et al., 2022) are simulated environments, so no real transaction or user account is touched, and SearchQA follows the Search-R1 retrieval protocol (Jin et al., 2025) over a public Wikipedia dump, which also contains no private or sensitive information.

REPRODUCIBILITY STATEMENT

We implement TITRATE on top of Verl's HybridFlow trainer (Sheng et al., 2025), including the skill-bank and controller packages, in Appendix C. Appendix B.2 provides the analyzer prompts and skill-update procedures, while Appendix B.5 presents the schedule derivations, assumptions, and the existing limitations. Benchmark protocols, hardware, and training settings are described in Appendix D, with benchmark-specific configurations summarized in Table 12. Section 4.1 defines the evaluation metrics, and Appendix G.1 reports repeated runs across four environment seeds on ALFWorld/Qwen3-1.7B. We save multiple versions of the configuration files and launcher scripts for the experimental settings. Our implementation records analyzer calls, available provider-reported token usage, and the applied distillation weight at each training step to support verification of extraction costs and adaptive scheduling. Stochastic policy rollouts and analyzer sampling can produce different skill text across repeated runs. Our experiments across four environment seeds on ALFWorld/Qwen3-1.7B show consistent final task performance in both evaluated distillation-weight ranges, despite variation in skill refresh counts.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/5be69a584901a26c521c2b51e40a4c20-Abstract-Conference.html.

Mohammad Sadegh Akhondzadeh, Vijay Lingam, Atula Tejaswi, Chanakya Ekbote, Sujay Sanghavi, and Aleksandar Bojchevski. Reward-gated on-policy distillation. *arXiv preprint arXiv:2607.04037*, 2026. URL https://arxiv.org/abs/2607.04037.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/64377.

Lawrence D Brown, T Tony Cai, and Anirban DasGupta. Interval estimation for a binomial proportion. *Statistical Science*, 16(2):101–133, 2001. doi: 10.1214/ss/1009213286. URL https://doi.org/10.1214/ss/1009213286.

Charles J Clopper and Egon S Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4):404–413, 1934. doi: 10.1093/biomet/26.4.404. URL https://doi.org/10.1093/biomet/26.4.404.

Databricks. Databricks AI runtime (AIR) CLI. https://docs.databricks.com/aws/en/machine-learning/ai-runtime/cli/, 2026. Accessed: 2026-09-14.

Yifan Ding, Xincheng Wei, Yoshua Y. Li, Ziheng Li, Yuquan Lu, Siyu Zhang, Dongsheng Ma, Rongxiang Weng, Xunliang Cai, and Yun Chen. SAF-OPD: Stable advantage fusion for on-policy distillation. *arXiv preprint arXiv:2607.29209*, 2026. URL https://arxiv.org/abs/2607.29209.

Yunshu Du, Wojciech M Czarnecki, Siddhant M Jayakumar, Mehrdad Farajtabar, Razvan Pascanu, and Balaji Lakshminarayanan. Adapting auxiliary losses using gradient similarity. *arXiv preprint arXiv:1812.02224*, 2018. URL https://arxiv.org/abs/1812.02224.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. URL https://jmlr.org/papers/v23/21-0998.html.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. OPTQ: Accurate quantization for generative pre-trained transformers. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=tcbBPnfwxS.

Andrew Gelman, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, 3rd edition, 2013. URL https://sites.stat.columbia.edu/gelman/book/.

Gemma Team. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024. URL https://arxiv.org/abs/2408.00118.

GLM-5-Team. Glm-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. MiniLLM: Knowledge distillation of large language models. In *International Conference on Learning Representations (ICLR)*, pp. 32694–32717, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/8ac015d409635f196f9e3e9dcfb9a94e-Abstract-Conference.html.

Haowei Guo, Baolong Bi, Ruicheng Zhang, Bingqian Sun, and Wentao Zhang. When should the teacher move? temporal coupling and stability in self on-policy distillation. *arXiv preprint arXiv:2606.03532*, 2026. URL https://arxiv.org/abs/2606.03532.

Zhuowen Han, Jinwei Xiao, Zhengxi Lu, Renren Jin, Zhiyuan Yao, Yuxin Liu, Hongyan Hao, Yueqing Sun, Yu Yang, Qi Gu, Xunliang Cai, and Deyi Xiong. Distill where you fail: Recovering learning signals of negative rl-groups from adaptive teacher guidance. *arXiv preprint arXiv:2608.00782*, 2026a. URL https://arxiv.org/abs/2608.00782.

Zihao Han, Tiangang Zhang, Huaibin Wang, and Yilun Sun. Adaptive teacher exposure for self-distillation in LLM reasoning. *arXiv preprint arXiv:2605.11458*, 2026b. URL https://arxiv.org/abs/2605.11458.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625. International Committee on Computational Linguistics, 2020. doi: 10.18653/v1/2020.coling-main.580. URL https://aclanthology.org/2020.coling-main.580/.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Jiazhen Huang, Xiao Chen, Xiao Luo, Yong Dai, Senkang Hu, and Yuzhi Zhao. Skill-conditioned gated self-distillation for LLM reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2026*, 2026. URL https://arxiv.org/abs/2605.28791. Accepted; proceedings forthcoming.

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. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/64121.

Hugging Face. Quantization overview. Transformers documentation, 2026. URL https://huggingface.co/docs/transformers/quantization/overview. Accessed September 19, 2026.

Sibgat Ul Islam, Jawad Ibn Ahad, Fuad Rahman, Mohammad Ruhul Amin, Nabeel Mohammed, and Shafin Rahman. Dynamic temperature scheduler for knowledge distillation. In *2025 IEEE International Conference on Big Data (BigData)*, pp. 1412–1417. IEEE, 2025. doi: 10.1109/BIGDATA66926.2025.11402644. URL https://doi.org/10.1109/BIGDATA66926.2025.11402644.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024. URL https://arxiv.org/abs/2401.04088.

Junguang Jiang, Baixu Chen, Junwei Pan, Ximei Wang, Dapeng Liu, Jie Jiang, and Mingsheng Long. ForkMerge: Mitigating negative transfer in auxiliary-task learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 30367–30389, 2023. doi: 10.52202/075280-1322. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/60f9118a849e8e9a0c67e2a36ad80ebf-Abstract-Conference.html.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, pp. 54107–54157, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/edac78c3e300629acfe6cbe9ca88fb84-Abstract-Conference.html.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Ö. Arık, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Second Conference on Language Modeling (COLM)*, 2025. URL https://openreview.net/forum?id=Rwhi91ideu.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611. Association for Computational Linguistics, 2017. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147/.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. VinePPO: Refining credit assignment in RL training of LLMs. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 29557–29590. PMLR, 2025. URL https://proceedings.mlr.press/v267/kazemnejad25a.html.

Jongwoo Ko, Sungnyun Kim, Tianyi Chen, and Se-Young Yun. DistiLLM: Towards streamlined distillation for large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 24872–24895. PMLR, 2024. URL https://proceedings.mlr.press/v235/ko24c.html.

Satyam Kumar and Saurabh Jha. CADENCE: Closing the reasoning gap via coverage-adaptive on-policy distillation. *arXiv preprint arXiv:2607.16955*, 2026. URL https://arxiv.org/abs/2607.16955.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026/.

Xingyu Lin, Harjatin Singh Baweja, George Kantor, and David Held. Adaptive auxiliary task weighting for reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/hash/0e900ad84f63618452210ab8baae0218-Abstract.html.

Haoran Liu, Yuwei Zhang, Xiyao Li, Bohan Lyu, and Jingbo Shang. HERO: Hindsight-enhanced reflection from environment observations for agentic self-distillation. *arXiv preprint arXiv:2606.11559*, 2026a. URL https://arxiv.org/abs/2606.11559.

Hongjun Liu, Yifei Ming, Shafiq Joty, and Chen Zhao. Harnessing LLM agents with skill programs. *arXiv preprint arXiv:2605.17734*, 2026b. URL https://arxiv.org/abs/2605.17734.

Xinyu Liu, Kechen Jiao, Chunyang Xiao, Runsong Zhao, Junhao Ruan, Bei Li, Jiahao Liu, Qifan Wang, Xin Chen, Jingang Wang, Chenglong Wang, Tong Xiao, and JingBo Zhu. Teacher-guided policy optimization for on-policy reasoning distillation under large policy divergence. *arXiv preprint arXiv:2605.13230*, 2026c. URL https://arxiv.org/abs/2605.13230.

Ye Liu, Srijan Bansal, Bo Pang, Yang Li, Zeyu Leo Liu, Yifei Ming, Zixuan Ke, Shafiq Joty, and Semih Yavuz. Procedural memory distillation: Online reflection for self-improving language models. *arXiv preprint arXiv:2607.01480*, 2026d. URL https://arxiv.org/abs/2607.01480.

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

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9802–9822. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.546. URL https://aclanthology.org/2023.acl-long.546/.

MiniMax. MiniMax-01: Scaling foundation models with lightning attention. *arXiv preprint arXiv:2501.08313*, 2025. URL https://arxiv.org/abs/2501.08313.

OpenAI. Introducing gpt-4.1 in the api. https://openai.com/index/gpt-4-1/, April 2025. Accessed: 2026-09-09.

Emiliano Penaloza, Dheeraj Vattikonda, Nicolas Gontier, Alexandre Lacoste, Laurent Charlin, and Massimo Caccia. Privileged information distillation for language models. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/62658.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-emnlp.378. URL https://aclanthology.org/2023.findings-emnlp.378/.

Daniel J Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A tutorial on Thompson sampling. *Foundations and Trends in Machine Learning*, 11(1):1–96, 2018. doi: 10.1561/2200000070. URL https://www.nowpublishers.com/article/Details/MAL-070.

Junhao Shen, Teng Zhang, Xiaoyan Zhao, and Hong Cheng. Dynamic skill lifecycle management for agentic reinforcement learning. *arXiv preprint arXiv:2605.10923*, 2026. URL https://arxiv.org/abs/2605.10923.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297. ACM, 2025. doi: 10.1145/3689031.3696075. URL https://doi.org/10.1145/3689031.3696075.

Yaorui Shi, Yuxin Chen, Zhengxi Lu, Yuchun Miao, Shugui Liu, Qi Gu, Xunliang Cai, Xiang Wang, and An Zhang. Skill1: Unified evolution of skill-augmented agents via reinforcement learning. *arXiv preprint arXiv:2605.06130*, 2026. URL https://arxiv.org/abs/2605.06130.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 8634–8652, 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn.

Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context. *arXiv preprint arXiv:2209.15189*, 2022. URL https://arxiv.org/abs/2209.15189.

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

Samuel Stanton, Pavel Izmailov, Polina Kirichenko, Alexander A. Alemi, and Andrew G. Wilson. Does knowledge distillation really work? In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 6906–6919, 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/hash/376c6b9ff3bedbbea56751a84fffc10c-Abstract.html.

Qitai Tan, Zefang Zong, Mo Li, Yipeng Shi, Yang Li, and Peng Chen. ATOD: Annealed turn-aware on-policy distillation for multi-turn agentic tasks. *arXiv preprint arXiv:2606.27814*, 2026. URL https://arxiv.org/abs/2606.27814.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. URL https://arxiv.org/abs/2302.13971.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. doi: 10.1162/tacl_a_00475. URL https://aclanthology.org/2022.tacl-1.31/.

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 (Volume 1: Long Papers)*, pp. 16022–16076. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

Songjun Tu, Chengdong Xu, Qichao Zhang, Yiwen Ma, Yaocheng Zhang, Linjing Li, Dong Li, Xiangyuan Lan, and Dongbin Zhao. UCOB: Learning to utilize and evolve agentic skills via credit-aware on-policy bidirectional self-distillation. *arXiv preprint arXiv:2606.29502*, 2026a. URL https://arxiv.org/abs/2606.29502.

Songjun Tu, Chengdong Xu, Qichao Zhang, Yaocheng Zhang, Xiangyuan Lan, Linjing Li, Dong Li, and Dongbin Zhao. Dynamic dual-granularity skill bank for agentic RL. *arXiv preprint arXiv:2603.28716*, 2026b. URL https://arxiv.org/abs/2603.28716.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research (TMLR)*, 2024a. URL https://openreview.net/forum?id=ehfRiF0R3a.

Hao Wang, Guozhi Wang, Han Xiao, Yufeng Zhou, Yue Pan, Jichao Wang, Ke Xu, Yafei Wen, Xiaohu Ruan, Xiaoxin Chen, and Honggang Qi. Skill-SD: Skill-conditioned self-distillation for multi-turn LLM agents. *arXiv preprint arXiv:2604.10674*, 2026. URL https://arxiv.org/abs/2604.10674.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022a. URL https://arxiv.org/abs/2212.03533.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439. Association for Computational Linguistics, 2024b. doi: 10.18653/v1/2024.acl-long.510. URL https://aclanthology.org/2024.acl-long.510/.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298. Association for Computational Linguistics, 2022b. doi: 10.18653/v1/2022.emnlp-main.775. URL https://aclanthology.org/2022.emnlp-main.775/.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025. URL https://proceedings.mlr.press/v267/wang25bx.html.

Edwin B Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927. doi: 10.1080/01621459.1927.10502953. URL https://doi.org/10.1080/01621459.1927.10502953.

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

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026. URL https://arxiv.org/abs/2602.08234.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. URL https://arxiv.org/abs/2412.15115.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, 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. *arXiv preprint arXiv:2604.03128*, 2026a. URL https://arxiv.org/abs/2604.03128.

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

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380. Association for Computational Linguistics, 2018. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, volume 35, pp. 20744–20757, 2022. doi: 10.52202/068431-1508. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/82ad13ec01f9fe44c01cb91814fd7b8c-Abstract-Conference.html.

Xiao Yu, Baolin Peng, Vineeth Vajipey, Hao Cheng, Michel Galley, Jianfeng Gao, and Zhou Yu. ExACT: Teaching AI agents to explore with Reflective-MCTS and exploratory learning. In *International Conference on Learning Representations*, volume 2025, pp. 65157–65184, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a3b893ba1de12f76020b03f7ae8e1afd-Abstract-Conference.html.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936. URL https://ojs.aaai.org/index.php/AAAI/article/view/29936.

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. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/64784.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *International Conference on Learning Representations (ICLR)*, pp. 19036–19066, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/52f050499cf82fa8efb588e263f6f3a7-Abstract-Conference.html.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, pp. 15585–15606, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/4410c0711e9154a7a2d26f9b3816d1ef-Abstract-Conference.html.

Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019. URL https://arxiv.org/abs/1909.08593.

A LIMITATIONS

Model and Training Constraints. Our RL experiments focus on small Qwen models from 1.7B to 7B. Generalization to other model families, such as Llama (Touvron et al., 2023), Google's Gemma (Gemma Team, 2024), MiniMax (MiniMax, 2025), and GLM (GLM-5-Team, 2026), larger parameter scales (14B+), LLMs using mixture-of-experts (MoE) architectures (Fedus et al., 2022; Jiang et al., 2024), quantization schemes (Hugging Face, 2026; Frantar et al., 2023), and LoRA training (Hu et al., 2022) is worth further investigations. We limit each training episode to at most 50 turns; performance on longer episodes remains to be evaluated. Also, due to budget limits, we didn't extend our ablation study to full benchmark validation across 3 models, which may reveal interesting cross-domain insights.

More Benchmarks TITRATE could be applied to other agentic benchmarks, including **ScienceWorld** (Wang et al., 2022b), **AppWorld** (Trivedi et al., 2024),**WebArena** (Zhou et al., 2024), τ^2**-bench** (Barres et al., 2026), and **SWE-bench** (Jimenez et al., 2024). Each extension requires a verifiable outcome reward and the identity fields used to construct task and class keys (§C.4). These keys must recur during training for skills to be reused, so their granularity should be chosen based on recurrence under each benchmark's prompt sampler(§E.1). For example, software engineering tasks could share skills at the repository level when individual issues do not recur. We leave these evaluations to future work.

Further Study for λ_t and Schedule Section 4.3 reveals that as the uplift of λ_t range, the performance of TITRATE improves but the performance gaps with one-off components (TI-BF, TI-TC, TI-CC, TI-SL, TI-NS2) remain yet reduces. This phenomenon suggests that in the higher value of λ_t range, it may be possible to simplify the design of TITRATE while maintaining its performance. How to further simplify its design and keep the performance-cost trade-off is an interesting future direction. Additionally, we also observed that not all λ_t range setting is suitable for all benchmarks, as the performance collapse in WebShop with the higher range of $\lambda_t \in [0.01, 0.1]$ setting discussed in Appendix G.2. This suggests that the λ_t range and schedule is sensitive on different benchmark, and a more adaptive approach should be explored in future work of TITRATE.

Further Study for REUSE/REFRESH Controller The current controller design applies simple probabilistic heuristics to determine when and how to refresh skills to avoid unnecessary complexity. However, a Beta belief from Monte Carlo continuations (Wang et al., 2024b; Kazemnejad et al., 2025) through a Beta-Binomial likelihood (Gelman et al., 2013; Russo et al., 2018) may be more suitable to estimate the probability of success rates for each skill on contrast lane considering the uncertainty. Similarly, the failure lane mechanism could be improved as well. We leave this probabilistic model improvements and more sophisticated controller designs to future work.

B METHOD DETAILS

B.1 SKILL KEY DETAILS

Motivation A skill key tells the analyzer which rollout trajectories to summarise together. As the basic unit of experience accumulation, a key must be stable across training steps, machine-independent, and derivable from environment metadata before extraction. Which granularity is *viable*, however, is decided by the environment's sampler that decides the recurrence frequency of key-related tasks. ALFWorld and WebShop redraw each step from a bounded pool, so an instance can be drawn again within the training horizon and a task-level key recurs often enough for the bank to do work (§E.1). In SearchQA each dataset row is an individual task and the dataloader walks a permutation of it, so no question can repeat until the epoch wraps — $1{,}325$ steps against a 300-step run — and task-level recurrence is zero by construction rather than merely low. A key that maps many instances onto one entry is therefore necessary in SearchQA.

Skill Derivation Our skill keys are accordingly derived from the agentic environment, at one of two granularities:

Table 3: Skill key derivation per benchmark.

Benchmark	Task identity	Class metadata	Depth / classes
ALFWorld	scenario file	task type	1 / 6
WebShop	goal instruction text	product category path	2 / 50-52
SearchQA	the question	data source + question type	2 / 16

1. **Task-level key**: the environment name followed by a hash of the normalised task identity concatenated by /. The raw task identity is a scenario file path or an entire question. Our normalisation applies unicode canonical composition, collapses whitespace, and strips a fixed punctuation set, since every SearchQA question always ends in a question mark.
2. **Class-level key**: the environment name and ordered benchmark metadata components, truncated to a configured depth and joined by /. Class keys are not hashed, as the vocabulary is small and the key appears in every metric and log line. Components are escaped — escape character, separator, then space in order.

Table 3 gives the key derivation for each benchmark. We set the depth per benchmark rather than sharing one value, because a class call only pools experience when at least two distinct instances of that class appear in the same batch, and how often that happens depends on how the benchmark's metadata branches relative to the number of instances a step draws. We therefore choose the depth maximising the expected number of such classes per step. Too fine a depth scatters a step's instances over many classes and few reach two — on ALFWorld, depth 2 yields 169 classes and 0.67 classes per step, against 6 and 4.64 at depth 1 — whereas too coarse a depth places most instances in one class, as depth 1 on WebShop does with 73% in a single category. Skill Bank also records its keyspace identity — environment, granularity, depth, derivation, and taxonomy fingerprint — and refuses to load under a mismatch.

Granularity Decision We use task-level keys for ALFWorld and WebShop because tasks recur during training, allowing skills and refresh decisions to remain specific to each task. SearchQA questions do not repeat within the 300-step or longer horizon, so we use class-level keys based on data source and question type to enable reuse across questions (Table 4). Class-level keys also reduce extraction costs on ALFWorld and WebShop and have similar success rates to task-level keys. §E.2 reports this trade-off.

Table 4: Key recurrence over 300 training steps and the class-level setting aligns with Table 3.

Benchmark	Task-level recurrence	Class-level recurrence	Selected key
ALFWorld	43.77%	99.64%	task
WebShop	12.23%	97.77%	task
SearchQA	0.00%	99.63%	class

B.2 Skill Creation & Update Details

B.2.1 Skill Initialization

When Initialization Occurs After collecting and scoring the current rollouts, we group them by skill key and look up each key in the bank. A key without a stored skill triggers initial extraction before teacher scoring. Such keys are excluded from $\mathcal{K}_t$ and $\mathcal{R}_t$ in Section 3.2: initialization is not subject to serving-history requirements, refresh admission predicates, or refresh quotas. Its analyzer calls are nevertheless included in the total extraction cost. This procedure also applies when all current rollouts succeed, since there is no stored skill to reuse.

Extracting and Storing the Initial Skill In the default contrastive configuration, the analyzer receives the selected current rollouts for the key, their task descriptions, and their observed outcomes, with no incumbent skill. It produces one shared skill: mixed outcomes use a contrastive instruction, while uniformly successful or failed rollouts use the corresponding single-outcome instruction

(Appendix B.2.2). If the analyzer completes successfully and returns nonempty skill text, the bank stores it as the first version for that skill key, recording the training step at creation and which rollout trajectories were used to generate it. The bank initializes the version's runtime statistics, and the skill conditions teacher scoring of the associated rollouts at the same training step. Measurements from this serving then populate the statistics used by later controller decisions.

Unsuccessful Initialization If extraction fails or returns no usable skill, the bank creates no entry for that key, and its rollouts receive no skill-based teacher supervision. If training continues, initialization is retried when the key next appears with analyzer input. Under the default failure guard, analyzer errors or unexpected empty outputs stop training when no usable skill is available anywhere in the step; a valid `NO_COMMON_PATTERN` response alone does not trigger this guard. In split mode, initialization is handled separately for each missing outcome-specific skill.

B.2.2 Skill Creation

Rollouts of one skill key can mix successful and failed trajectories. The analyzer have two ways to extract a skill from them:

1. **Split extraction** (Figures 3 and 4): Each key can hold two skills, extracted separately from successful and failed trajectories. During teacher re-scoring, each trajectory receives only the skill corresponding to its observed outcome. The skills are stored separately, although refresh decisions can use evidence pooled across the pair. This option is enabled by `algorithm.seed.bank.outcome_split=true`.
2. **Contrastive extraction** (Figure 5): with `outcome_split=false`, each key holds a single skill extracted from both successful and failed trajectories together. When both outcomes are present, the prompt asks what distinguishes successful attempts from failed ones; otherwise, it uses a single-outcome instruction. We use this unsplit configuration in our experiments.

Both task-level and class-level keys can use either extraction method. The only difference is that task-level keys are extracted from multiple instances of the same task, whereas class-level keys are extracted from multiple instances of different tasks. Through our experiments, we find contrastive extraction is more effective than split extraction, as it allows the analyzer to learn from both successful and failed trajectories together, leading to a more comprehensive understanding of the task similar to the finding from prior work (Yu et al., 2025). Therefore, we use **contrastive extraction as the default option** in our experiments.

B.2.3 Incremental Update

Skill Update via Revision To limit unnecessary changes to the teacher's privileged context, `algorithm.seed.bank.skill_evolution=true` asks the analyzer to revise the existing skill using new trajectories (Figure 6). The placeholder `{incumbent}` contains the stored skill text for the key, or its selected outcome-specific skill in split mode. This text is placed after the new trajectories, at the end of the analyzer's prompt. The placeholder `{branch_instruction}` contains the extraction instruction selected for the granularity and outcome mix: for example, identifying a successful workflow, examining failed attempts, or contrasting successes with failures. It is inserted verbatim into the revision prompt to specify what to learn from the new evidence. The analyzer preserves uncontradicted guidance, corrects contradicted instructions, incorporates new evidence, and returns the complete revised skill. Each revision records the previous skill actually supplied to the analyzer. We also monitor the ratio of new to previous skill length to detect excessive growth across updates.

Safeguards for Skill Statistics To measure how a skill changes the teacher's scores, we compare teacher and student probabilities for the *same sampled response*. The teacher receives the skill-augmented prompt and scores the response already generated by the student; it does not generate a new response for this comparison. We also reuse the response mask, which identifies the token positions included in the comparison. The implementation applies four checks:

```
Analyze the following group of agent episodes and return ONLY valid JSON.

You need to complete one field:
1. These {total} episodes are independent attempts at the SAME task, and all of them
     succeeded.
Write one episode_skill that extracts them into workflow: the core decision rule and action
ordering common to them. Name actions in the terms they were actually taken ("go to desk 1",
"take pencil 2"), never as abstracted paraphrase ("locate", "acquire", "search surfaces").

Important constraints:
- The episodes are attempts at one task; write ONE skill for the task, not one per episode.
- Do not mention the episodes at all -- not by number, position, or the word "episode": the
skill is consumed where these episodes are not visible. Write about the task and the actions
     to
take.
- episode_skill MUST begin with the exact prefix "ACTION: " followed by the skill itself. The
prefix is a protocol marker and will be stripped.
- Write episode_skill as one short policy-facing skill, not as retrospective explanation of
     the
trajectory.
- Return only this top-level field: episode_skill.

Return format:
{
  "episode_skill": "ACTION: string"
}

Episode context:
Task: {task_description}
Outcomes: {total} succeeded, 0 failed, of {total} attempts shown
Episodes:
{formatted episode group}
```

Figure 3: **Split skill creation: success prompt.** Complete task-level creation prompt (TASK_SUCCESS), with runtime data replaced by placeholders and the optional summary field disabled. Only successful trajectories are included in the episode group.

1. **Compare matching token positions.** Teacher scores, student scores, entropy values, and the response mask must have matching dimensions. The code raises an error for mismatched dimensions or non-finite values in the scored tokens, rather than silently omitting invalid measurements.
2. **Record the precision used.** The test of whether the distillation gate exceeds 0.5 uses the scores' original numerical precision. We record the actual data type because rounding can change the result near this threshold.
3. **Measure before updating the policy.** Skill statistics use teacher scores and saved student scores from before the policy update. This prevents a change caused by optimization from being attributed to the skill.
4. **Keep each skill version and its statistics together.** A successful refresh replaces the stored skill and resets its version-specific statistics together, so measurements from the old skill are not assigned to the new one. If the refresh fails, the existing skill and its age are retained.

B.3 Runtime Statistics in Skill Bank

The skill bank's runtime statistics support both **the adaptive distillation-weight scheduler(Component A)** and the **Reuse/Refresh controller(Component B)**. We first establish shared notation, then describe the per-key, version-scoped state and the inputs used by each component.

The skill bank stores and exposes runtime statistics through three views: **a read-only copy of each skill's statistics for both components** to read at a training step, **per-step scalar metrics** in the run log, and **an append-only analytics stream** with one row per (step, skill key) for analysis-only purposes. The underlying accumulators maintain basic statistics, such as counts and sums—for example, successful and total rollouts. Derived metrics, such as success rates, are computed from these accumulators when the read-only view is constructed. The active accumulators summarize measurements collected since the current skill version was created. Meanwhile, the append-only

```
Analyze the following group of agent episodes and return ONLY valid JSON.

You need to complete one field:
1. These {total} episodes are independent attempts at the SAME task, and all of them failed.
Write one episode_skill that does two things in order: first name the action pattern that did
not advance the task, in the terms it actually took; then name the actions in these attempts
that DID make progress before that point. Describe only what is visible in the attempts. Name
actions in the terms they were actually taken ("go to desk 1", "take pencil 2"), never as
abstracted paraphrase ("locate", "acquire", "search surfaces").

Important constraints:
- The episodes are attempts at one task; write ONE skill for the task, not one per episode.
- Do not mention the episodes at all -- not by number, position, or the word "episode": the
skill is consumed where these episodes are not visible. Write about the task and the actions
    to
take.
- episode_skill MUST begin with the exact prefix "ACTION: " followed by the skill itself. The
prefix is a protocol marker and will be stripped.
- Write episode_skill as one short policy-facing skill, not as retrospective explanation of
    the
trajectory.
- Return only this top-level field: episode_skill.

Return format:
{
  "episode_skill": "ACTION: string"
}

Episode context:
Task: {task_description}
Outcomes: 0 succeeded, {total} failed, of {total} attempts shown
Episodes:
{formatted episode group}
```

Figure 4: **Split skill creation: failure prompt.** Complete task-level creation prompt (`TASK_FAILURE`), with runtime data replaced by placeholders and the optional summary field disabled. Only failed trajectories are included in the episode group.

analytics stream retains historical measurements from individual training steps, including those collected under earlier skill versions, for later analysis.

Notation Table 5 summarizes the notation used by both components. For pre-update measurements, define $\mathrm{gap}_{r,i} = \ell^{\mathrm{teach}}_{r,i} - \ell^{\mathrm{old}}_{r,i}$ for token i in row r, with gate slope $\beta_{\mathrm{OPD}} = 5$. The gate $\omega_{r,i}$ is evaluated at $\theta = \theta_{\mathrm{old}}$ for these measurements; the OPD loss recomputes it against the current policy during optimization, with its value detached from differentiation (Equation 30).

Table 5: Notation for the adaptive distillation-weight scheduler (Component A; Section 3.3) and the REUSE/REFRESH controller (Component B; Section 3.2).

Symbol	Meaning	Component
SR_t	batch success rate, the schedule's driver	A
ς_t	disjoint-window noise scale	A
$\sigma(\cdot)$	the logistic, as in the gate $\sigma(\beta_{\mathrm{OPD}}\,\mathrm{gap}_{r,i})$	A
$s_k(t),\ m_k(t)$	successful / served rollouts of key k	B
$\mathbf{u}_k(t)$	the serving tuple (s_k, m_k, n_k, d_k)	B

Per-Skill-Key, Version-Scoped State Each bank entry maintains runtime statistics for its current skill version. A successful refresh resets these active statistics for the new version; previously logged per-step measurements remain available for historical analysis. Cumulative extraction cost is maintained across all versions of the same key and is not reset.

When either component requests an entry's statistics, the skill bank computes derived values from the stored accumulators and version metadata to construct the read-only view. These values are calculated at request time. For skill key k at training step t, the following statistics summarize measurements accumulated under its current skill version and the version's age:

```
Analyze the following group of agent episodes and return ONLY valid JSON.

You need to complete one field:
1. These {total} episodes are independent attempts at the SAME task; {n_succeeded} succeeded
     and
{n_failed} failed. Write one episode_skill capturing what the successful attempts did
differently from the failed ones -- the decision or action ordering that separates them.
     Ground
it in the contrast, not in either group alone.

Important constraints:
- The episodes are attempts at one task; write ONE skill for the task, not one per episode.
- Do not mention the episodes at all -- not by number, position, or the word "episode": the
skill is consumed where these episodes are not visible. Write about the task and the actions
     to
take.
- episode_skill MUST begin with the exact prefix "ACTION: " followed by the skill itself. The
prefix is a protocol marker and will be stripped.
- Write episode_skill as one short policy-facing skill, not as retrospective explanation of
     the
trajectory.
- Return only this top-level field: episode_skill.

Return format:
{
  "episode_skill": "ACTION: string"
}

Episode context:
Task: {task_description}
Outcomes: {n_succeeded} succeeded, {n_failed} failed, of {total} attempts shown
Episodes:
{formatted episode group}
```

Figure 5: **Contrastive skill creation** with `outcome_split=false`. Complete creation prompt for mixed outcomes, with runtime data replaced by placeholders. Single-outcome groups use a successful experience summary or avoidance instruction for failed attempts instead. The class-level version additionally imposes role-based and cross-instance constraints and permits `NO_COMMON_PATTERN`.

Table 6: Per-entry state. *Exposure* answers whether a version has had a fair trial, *trust* whether it is sustained by fresh confirmation or by the absence of it, and *value* carries the loss-native accumulators used to compute derived statistics when requested.

Block	Contents	Reset?
exposure	steps served; teacher-active tokens	yes
trust	failed refreshes and no-common-pattern returns, both since the last write	yes
value	25 accumulators: counts, first and second moments of the per-token gap, the gate level and the gate event, the trend cross-products, and their failure-stratified siblings	yes
birth	the same accumulators over the version's first served step, sealed once	yes
cost	cumulative analyzer tokens charged to the key	**no**

- **Mean teacher–student gap,** $\bar{\mathrm{gap}}_k(t)$**:** the average per-token log-probability difference between the teacher and student over observations collected under this version.
- **Gap trend,** $b_{\mathrm{gap},k}(t)$**:** the ordinary-least-squares slope of the observed gap against training step, using observations collected since this version was created.
- **Success rate,** $\mathrm{SR}_k^{\mathrm{active}}(t)$**:** the fraction of observed rollouts that succeeded while this version was active.
- **Mean gate level,** $\bar{\omega}_k(t)$**:** the average distillation gate value $\omega_{r,i} = \sigma(\beta_{\mathrm{OPD}}\,\mathrm{gap}_{r,i})$ over observations collected under this version.
- **Gate variance,** $\mathrm{Var}_k(\omega; t)$**:** the variation in gate values, computed from their accumulated first and second moments.

```
The selected extraction instruction (split or unsplit) is embedded VERBATIM as
{branch_instruction}. The stored skill text replaces {incumbent} in the current-skill block,
which follows the new episode transcripts at the end of the prompt.

=== revision frame, wrapping the numbered skill instruction ===
A current skill for this task is shown below under "Current skill". Your job is to produce the NEXT
VERSION of that skill, not a new skill written from scratch.
   What these episodes add -- read this as a description of the evidence to fold in, not as a demand
to start over: {branch_instruction}
   How to revise: sort every part of the current skill into exactly one of three cases.
(1) CONTRADICTED by these episodes -- delete it or replace it with what the episodes show instead;
never keep it, and never merely soften or qualify it. (2) NOT MENTIONED by these episodes -- keep it,
word for word where possible; if genuinely new evidence needs space you may consolidate it, but never
lose its actionable meaning and never drop it merely because it is unrelated. (3) SUPPORTED by these
episodes -- keep it, and sharpen it with what they newly show. The evidence description above says
what to TAKE FROM these episodes; any constraint there to describe only what is visible in the
attempts governs what you ADD, never what you KEEP. Return the COMPLETE revised skill in
episode_skill -- the whole skill as it should now stand, not a diff, not a list of changes, and not a
commentary on what you altered.
   Length: the revised skill must be about as long as the current skill and must not grow with each
revision. This is the next version of ONE short skill, not an accumulation of everything ever
learned. When new evidence needs room, FIRST merge overlapping guidance, THEN compress uncontradicted
guidance without deleting its actionable meaning, or remove what these episodes have shown to be
WRONG. Never delete guidance merely because it is unrelated to these episodes. If something in the
current skill is now known to be wrong, remove it instead of qualifying it.

=== current-skill block, appended after the episode transcripts ===
Current skill for this key -- THIS IS WHAT YOU ARE REVISING. Go through it one instruction at a time
and decide each one's fate, then write the result. DELETE an instruction only if these episodes
CONTRADICT it -- and then it must not survive in any form, not reworded and not qualified. KEEP every
other instruction, including ones added by earlier revisions and ones about situations these episodes
never reach: being unrelated to this batch is never a reason to drop something. Then ADD what these
episodes newly show. Your answer must still contain every instruction below that these episodes do
not contradict.
---BEGIN CURRENT SKILL---
{incumbent}
---END CURRENT SKILL---
```

Figure 6: **Incremental update**. The revision frame embeds the selected extraction instruction as `{branch_instruction}`. The to-be-updated skill replaces `{incumbent}` in the final current-skill block. The analyzer returns the complete revised skill.

- **Age in training steps,** $a_k^{\text{step}}(t)$**:** the number of training steps elapsed since this version was written to the bank.
- **Age in model versions,** $a_k^{\text{model}}(t)$**:** the difference between the current model-version counter and its value when this skill version was written.

Both age measures start from the last *successful* skill write. A failed refresh leaves the current version in place and does not reset its age.

Inputs to the Adaptive Distillation-Weight Scheduler The inputs follow the two stages defined in Section 3.3. **Stage 1: progress-aware estimation** uses training history available before step t to compute the base distillation weight λ_t^{raw}. **Stage 2: downward-only correction** uses the current step's RL and OPD signals, available after the REUSE/REFRESH decision and teacher scoring, to adjust λ_t^{raw} and obtain λ_t.

Stage 1 uses batch success-rate history through step $t-1$, allowing λ_t^{raw} to be computed before the current step's REUSE/REFRESH decision. Stage 2 computes gradient alignment and magnitudes using the token masks and normalization weights of the corresponding RL and OPD losses. For the token-mean reduction in the displayed loss equations, $w_{D,r,i} = 1/Z_M$ and $w_{R,r,i} = 1/Z_m$. These weights normalize each microbatch's losses; trainer gradient-accumulation scaling is applied separately. Omitting these weights can change whether the two gradients appear aligned or opposed.

Inputs to the Reuse/Refresh Controller For skill key k at training step t, the bank provides current rollout counts $(s_k(t), m_k(t))$ and completed-serving history $H_k(t)$ to the controller defined in Section 3.2. The history includes group counts n_k and advantage-flat group counts d_k, recorded using the tests detailed in Section B.4.1. These statistics summarize the key's rollout outcomes and the variation in rollout-mean advantages in previous servings.

Additional Records and Their Uses Beyond supplying controller inputs, the bank records information for historical analysis and run monitoring:

Table 7: Inputs to the adaptive distillation-weight scheduler (Component A). The time reference identifies the training steps used to determine λ_t. Stage 1 uses training history through step $t-1$ to determine the base weight. Stage 2 uses measurements from step t, available after teacher scoring and advantage computation, to determine the final weight before the policy update.

Symbol	Definition	Used by	Time reference
$\{\mathrm{SR}_j\}_{j<t}$	batch success-rate history over full rollout batches	Stage 1	$\leq t-1$
Δ_{t-1}	split-half drift of SR over a lookback of H	Stage 1	$\leq t-1$
ς_{t-1}	noise scale from differences aged out of that window	Stage 1	$\leq t-1$
v_t	the progress clock, a float	Stage 1	State at step t
$\mathrm{gap}_{r,i}$	per-token teacher minus pre-update student log-probability	Stage 2	t
$\omega_{r,i}$	$\sigma(\beta_{\mathrm{OPD}}\,\mathrm{gap}_{r,i})$, the detached pre-update gate	Stage 2	t
$A_{r,i}$	per-token advantage	Stage 2	t
$M_{r,i}$, $m_{r,i}$	the OPD and RL token masks	Stage 2	t
$w_{D,r,i}$, $w_{R,r,i}$	per-token inverse-normalisation weights, resolved per gradient	Stage 2	t
N_D, N_R	$\|\omega_{r,i}M_{r,i}w_{D,r,i}\|_2$ and $\|A_{r,i}m_{r,i}w_{R,r,i}\|_2$	Stage 2	t
q_t	the undone fraction, from those same weighted vectors	Stage 2	t
entropy	mean pre-update policy entropy on the scored tokens	override	t

Table 8: Component B's inputs. Superscripts (0) and (-1) denote the current and last completed servings. Current outcome counts are available before the refresh decision; the current advantage census is not yet available.

Symbol	Definition	Used by	Lag
$m_k(t)$	rollouts served for key k	lane partition; both lanes	0
$s_k(t)$	successful rollouts, $0 \leq s_k \leq m_k$	lane partition; both lanes	0
$n_k(t)$	advantage groups served for k	FAILURE admission; rank	1
$d_k(t)$	of those, groups that are advantage-flat	FAILURE admission; rank	1
$H_k(t)$	the key's completed servings, in time order	admission history	—
$p^{(j)}$	$s^{(j)}/m^{(j)}$, the per-serving success rate	CONTRAST no-improvement	—
$\hat{\rho}$	$d^{(-1)}/n^{(-1)}$, the advantage-flat group fraction	FAILURE rank	1
E_k	remaining failure mass, Eq. 13; 0 unless $0 < s < m$	CONTRAST admission and rank	0

- **Skill exposure:** steps served and teacher-active token counts measure how much a skill version has been used and how many tokens received teacher guidance.
- **Skill evolution:** initial measurements, subsequent teacher–student gap and gate statistics, and separate statistics for failed rollouts support comparisons between a version's initial and later behavior, including on unsuccessful attempts.
- **Refresh outcomes and cost:** successful writes, failed refreshes, no-common-pattern returns, and cumulative analyzer-token usage support analysis of refresh frequency, unsuccessful update attempts, and extraction cost.
- **Version provenance:** skill text, content identifiers, creation steps, and training-branch identifiers associate measurements with the skill version that produced them and distinguish records from different training restarts.
- **Bank monitoring:** bank size, entry-age summaries, and analytics-write failure counts track bank growth, how long versions remain in use, and whether historical measurements were successfully recorded.

Checkpointing and Recovery The skill bank and its accumulated statistics are saved with the policy checkpoint. When training restarts from that checkpoint, the policy, skill versions, and corresponding statistics are restored together.

B.4 REUSE/REFRESH CONTROLLER: ADDITIONAL DETAILS

In this section, we follow the notation and admission rules of Section 3.2 and provide the evidence-collection details, history boundary cases, statistical derivations, and quota accounting needed to reproduce the controller.

Evidence Population and Availability Controller statistics cover all rollouts associated with a skill key, including trajectories omitted from the analyzer prompt. For example, a class key covering 4 prompts with 8 rollouts each contributes 4 groups and 32 rollouts, even if the analyzer receives only a subset of those trajectories.

At the refresh decision, current rollout outcomes are available, but current advantages have not yet been computed. The runtime diagnostic and skill key priority therefore read **the preceding completed serving's** $(d_k^{(-1)}, n_k^{(-1)})$, while **current success counts** can enter admission immediately. The new group statistics become available to later decisions only **after the training step commits**. This aligns with how our 2 stage cosine schedule leverages the historical and current OPD/RL statistics. Appendix C.1 gives the execution and commit order.

Serving History and Boundary Cases The controller's per-key serving history persists across successful skill refreshes, retaining the recent measurable servings required by the admission windows. It can therefore span skill versions, unlike the version-scoped runtime accumulators, which reset after a successful write. A serving is one occurrence of a key in a training batch hosted in the in-memory service of skill bank. History advances only for measurable servings of that key; intervening steps where it is absent do not advance its history. **If the requested window is unavailable, the controller refuses admission rather than substituting a shorter window**. Every contrast comparison also requires positive rollout counts throughout its window; a zero count prevents admission.

The default horizons are $L_{\text{fail}} = L_{\text{cont}} = 2$. The implementation also accepts $L_\ell = 1$, but this gives the same admission decisions as $L_\ell = 2$: Failure still requires every group in the preceding census to be advantage-flat, which already implies failure at that serving, and Contrast still compares two successive servings. Thus setting $L_\ell = 1$ does not remove the historical-evidence requirement.

B.4.1 ADVANTAGE-FLAT GROUP TESTS

Advantage Calculation A multi-turn rollout contributes one training row per turn. Let $\mathcal{B}_g$ contain all training rows from rollouts in prompt group g, and let R_r be the summed token reward in row r. For $|\mathcal{B}_g| \geq 2$, the group mean, sample standard deviation, and normalized advantage used by ALFWorld and SearchQA are

$$\mu_g = \frac{1}{|\mathcal{B}_g|} \sum_{r \in \mathcal{B}_g} R_r, \qquad \sigma_g = \sqrt{\frac{1}{|\mathcal{B}_g| - 1} \sum_{r \in \mathcal{B}_g} (R_r - \mu_g)^2},$$
$$A_r^{\text{out}} = \frac{R_r - \mu_g}{\sigma_g + \varepsilon}, \qquad \varepsilon = 10^{-6}. \tag{11}$$

WebShop uses $A_r^{\text{out}} = R_r - \mu_g$. A group containing only one row instead uses $(\mu_g, \sigma_g) = (0, 1)$. With the additional advantage terms disabled in these experiments, the row scalar is broadcast to its response tokens as $A_{r,j} = A_r^{\text{out}} \, \text{mask}_{r,j}$, where the mask excludes padding.

Rollout Means and Group Test The controller averages advantages over all response tokens in each rollout, pooling its turn rows before dividing:

$$\bar{A}_\tau = \frac{\sum_{r \in \tau} \sum_j A_{r,j} \, \text{mask}_{r,j}}{\sum_{r \in \tau} \sum_j \text{mask}_{r,j}}. \tag{12}$$

This weights turns by response length. Equation 1 then tests the spread of these rollout means within each prompt group, with tolerance 10^{-8}, and requires every rollout in the group to have failed. Groups are tested separately even when they share a class-level skill key; d_k counts the groups that pass. A rollout without response tokens has an undefined mean. The census compares available means and requires at least one; a group with no available mean is not counted as advantage-flat.

Interpretation and Limits Identical row scores in a group with at least two rows yield zero centered advantages. The test excludes uniformly successful groups, so $d_k = n_k > 0$ implies $s_k = 0$. The converse need not hold: failed rollouts can have different advantages, for example because of invalid-action penalties.

The diagnostic concerns rollout means, not every token's advantage. Equal means can hide variation across turns, and a single available mean has zero spread by construction. Thus passing the test does not generally certify a zero policy gradient, particularly for singleton or incompletely observed groups. This is the mean-based diagnostic used by the controller; it does not test the per-token advantage extrema.

B.4.2 Remaining Failure Mass for Contrast Admission

Purpose and Definition The contrast rule requires more than stalled mixed outcomes: With the same sample size, a key with 7 successes in 8 rollouts presents less observed failure than one with a single success. The statistic E_k discounts the empirical failure rate when evidence is thin. It filters contrast keys through $E_k > \tau_{\mathcal{U}}$ together with the persistence condition; every admitted contrast key refreshes in the reported experiments. It measures remaining failure evidence, not the expected improvement from refreshing a skill.

Write $s = s_k^{(0)}$, $m = m_k^{(0)}$, and $\hat{p} = s/m$. For $0 < s < m$, the implemented statistic is

$$E_k(t) = 1 - p_{\mathrm{W}}^{\mathrm{hi}}, \qquad p_{\mathrm{W}}^{\mathrm{hi}} = \frac{\hat{p} + \dfrac{z^2}{2m} + z\sqrt{\dfrac{\hat{p}(1-\hat{p})}{m} + \dfrac{z^2}{4m^2}}}{1 + \dfrac{z^2}{m}}. \tag{13}$$

The implementation returns $E_k = 0$ outside the mixed-outcome lane.

Derivation and Statistical Interpretation Under independent Bernoulli outcomes with common success probability p, let $S \sim \mathrm{Bin}(m, p)$. Inverting the normal score test for a hypothesized p_0 gives the upper boundary

$$\hat{p} - p_0 = -z\sqrt{\frac{p_0(1-p_0)}{m}}, \qquad z = \Phi^{-1}(1-\alpha/2). \tag{14}$$

Squaring yields $(1 + z^2/m)p_0^2 - (2\hat{p} + z^2/m)p_0 + \hat{p}^2 = 0$; its larger root is the Wilson upper bound (Wilson, 1927) in Equation 13. Complementing it gives an approximate lower confidence bound on $1 - p$. At fixed m, E_k decreases with s; at fixed interior $\hat{p}$, it increases with m. Thus **the same empirical failure rate receives greater weight when supported by more rollouts**.

The confidence interpretation assumes independent outcomes with a common probability. At task granularity this is a working model for repeated rollouts under a fixed policy. Class keys pool tasks of different difficulty and can introduce dependence among outcomes. We therefore use E_k as **an evidence-weighted admission statistic at class granularity**. Neither this interval nor the admission rule establishes that skill revision will improve performance.

Comparison with Other Intervals For reference, the exact Clopper–Pearson upper endpoint (Clopper & Pearson, 1934) for mixed outcomes is

$$p_{\mathrm{CP}}^{\mathrm{hi}} = B^{-1}(1-\alpha/2; s+1, m-s), \tag{15}$$

where B^{-1} is a Beta quantile. Wilson avoids numerical quantile inversion and gives a simple analytic score. The Wald alternative uses $\hat{p} + z\sqrt{\hat{p}(1-\hat{p})/m}$; it can exceed one and has zero width at empirical rates zero and one (Brown et al., 2001). Table 9 compares the implied failure scores. For example, clipping Wald at zero makes $6/8$ and $7/8$ indistinguishable, while Wilson preserves their order. Wilson and Clopper–Pearson need not make identical admission decisions at small sample sizes.

Parameter Choices We use $z = \Phi^{-1}(0.975) \simeq 1.96$, corresponding to the upper endpoint of a nominal two-sided 95% Wilson interval; its complement has nominal one-sided level 97.5% under

Table 9: Remaining failure mass $1 - p^{\mathrm{hi}}$ under the three bounds at $z = 1.96$. Top: every mixed outcome at the task-granularity group size $m = 8$; the Wald value is clipped at zero where its upper endpoint exceeds one. Bottom: $\hat{p} = 1/2$ as the group size grows, the regime of class granularity. Wilson and Clopper–Pearson converge as m grows; Wald degenerates at the high-success end.

s/m	$\hat{p}$	Wilson $E_k(t)$	Clopper–Pearson	Wald
1/8	0.125	0.529	0.473	0.646
2/8	0.250	0.409	0.349	0.450
3/8	0.375	0.306	0.245	0.290
4/8	0.500	0.215	0.157	0.154
5/8	0.625	0.137	0.085	0.040
6/8	0.750	0.071	0.032	0.000
7/8	0.875	0.022	0.003	0.000
2/4	0.500	0.150	0.068	0.010
8/16	0.500	0.280	0.247	0.255
16/32	0.500	0.336	0.319	0.327
32/64	0.500	0.381	0.372	0.378
64/128	0.500	0.415	0.410	0.413

the working model. **Increasing z lowers E_k and makes admission more conservative at fixed $\tau_{\mathcal{U}}$.** For fixed m, changing z preserves the success-count ordering, but it can change threshold decisions. **With varying m, it can also change the ordering across keys**. This aligns with our supports for task-level and class-level keys: a larger m can give more weight to the same empirical failure rate.

The default $\tau_{\mathcal{U}} = 0.05$ permits mixed outcomes $s \in \{1, \ldots, 6\}$ at $m = 8$ and excludes $7/8$, subject to the history condition. The TI-TC ablation uses $\tau_{\mathcal{U}} = 0.2152$, permitting $s \in \{1, \ldots, 4\}$ at that sample size. Admission uses a strict inequality: $E(4, 8) = 0.21521\ldots$ exceeds the rounded constant 0.2152, whereas setting the threshold to the exact $E(4, 8)$ would exclude that point. At other sample sizes the same threshold need not correspond to the same empirical success-rate cutoff. The table makes these operating points explicit.

B.4.3 Failure Ranking and Contrast Filtering

Let $\mathcal{A}_t^{\ell} = \{k \in \mathcal{K}_t^{\ell} : \mathrm{Adm}_k^{\ell}(t)\}$ denote the keys admitted by the lane-specific predicates in Section 3.2.

Failure-score Reduction The implementation evaluates a confidence-inspired score for the advantage-flat group fraction $\hat{\rho} = d^{(-1)}/n^{(-1)}$. For independent group indicators with common mean ρ, Hoeffding's inequality gives

$$\Pr(\hat{\rho} - \rho \geq \epsilon) \leq e^{-2n^{(-1)}\epsilon^2}. \tag{16}$$

Assigning $\delta_n = \alpha/[n(n+1)]$ yields the implemented expression

$$\widetilde{U}_k^{\mathrm{FAILURE}} = \hat{\rho} - w(n^{(-1)}), \qquad w(n) = \sqrt{\frac{\ln(n(n+1)/\alpha)}{2n}}. \tag{17}$$

This construction does not provide simultaneous coverage across successive servings: each serving supplies a new sample, and the allocation is over sample sizes, not training steps. The ranking equivalence below is algebraic and does not require a coverage claim.

Failure admission reads the same census and requires

$$k \in \mathcal{A}_t^{\textsc{Failure}} \implies d_k^{(-1)} = n_k^{(-1)} > 0 \implies \hat{\rho} = 1. \tag{18}$$

Consequently,

$$\widetilde{U}_k^{\mathrm{FAILURE}} = 1 - w(n_k^{(-1)}) \quad \text{on } \mathcal{A}_t^{\textsc{Failure}}. \tag{19}$$

To establish the ordering, extend $w(n)$ to real $n \geq 1$ and differentiate its square:

$$\frac{\mathrm{d}}{\mathrm{d}n} w(n)^2 = \frac{1 + \dfrac{n}{n+1} - \ln\big(n(n+1)/\alpha\big)}{2n^2} < 0 \qquad (n \geq 1,\ \alpha = 0.05). \tag{20}$$

Table 10: The FAILURE utility on the admitted set. The implemented bound (Equation 19) is a strictly increasing function of the group count, hence rank-equivalent to the group count itself (Equation 21). Values at $\alpha = 0.05$.

$n^{(-1)}$	1	2	3	4	8	20
$\widetilde{U}_k^{\mathrm{FAILURE}}(t) = 1 - w(n^{(-1)})$	-0.358	-0.094	$+0.044$	$+0.135$	$+0.326$	$+0.525$
$U_k^{\mathrm{FAILURE}}(t) = n^{(-1)}$	1	2	3	4	8	20

The inequality follows because $1 + n/(n+1) < 2$, whereas $\ln(n(n+1)/0.05) \geq \ln 40 > 2$. Hence $w(n)$ strictly decreases and $1 - w(n)$ strictly increases with n. For admitted keys, descending order of the implemented score is therefore exactly descending order of $n_k^{(-1)}$:

$$\mathrm{rank}_{\mathrm{fail}}(\widetilde{U}^{\mathrm{FAILURE}}) = \mathrm{rank}_{\mathrm{fail}}(U^{\mathrm{FAILURE}}), \qquad U_k^{\mathrm{FAILURE}} = n_k^{(-1)}. \tag{21}$$

This is an order-preserving simplification: our implementation initially retains the confidence-inspired score, then adapts the group count to express the same priority. At the configured $\alpha = 0.05$, the parameter affects score magnitudes but need not appear in the ranking rule. Equal group counts also give equal implemented scores, so the same seeded tie-break preserves the selected set. Table 10 illustrates the reduction. Negative implemented scores are retained rather than clipped; clipping would create additional ties.

The main-text quantity $\beta_{\mathrm{FAILURE}}|\mathcal{K}_t^{\mathrm{FAILURE}}|$ denotes newly accrued allowance, not a per-step spending limit; actual selections also use carried allowance. For contrast keys, the ordering described in the main text does not affect membership in the refresh set in the reported experiments: every admitted key refreshes.

Failure-lane Ranking Only admitted failure keys compete for the failure refresh quota. We rank these keys by decreasing priority, then increasing tie-break hash $\mathrm{tb}(k)$, the SHA-256 hash of the run seed and key. For $k \in \mathcal{A}_t^{\mathrm{FAILURE}}$, define

$$\begin{aligned}\mathrm{rank}_{\mathrm{fail}}(k) = 1 + \big|\{k' \in \mathcal{A}_t^{\mathrm{FAILURE}} : U_{k'}^{\mathrm{FAILURE}} > U_k^{\mathrm{FAILURE}} \text{ or} \\ [U_{k'}^{\mathrm{FAILURE}} = U_k^{\mathrm{FAILURE}} \ \wedge\ \mathrm{tb}(k') < \mathrm{tb}(k)]\}\big|.\end{aligned} \tag{22}$$

The time argument t of U is omitted for brevity.

Failure Quota The rate $\beta_{\mathrm{FAILURE}} \in [0, 1]$ controls how much refresh allowance accrues per candidate failure key. Lower rates restrict refreshes to higher-ranked admitted keys when the available allowance is insufficient; unused allowance carries forward.

Let b_{t-1} be the failure lane's unused allowance before step t, with $b_0 = 0$. Each candidate failure key, counted before admission, contributes β_{FAILURE} calls. The number of selected failure refreshes and the remaining allowance are

$$\begin{aligned} q_t^{\mathrm{FAILURE}} &= \min\big(|\mathcal{A}_t^{\mathrm{FAILURE}}|, \lfloor b_{t-1} + \beta_{\mathrm{FAILURE}}|\mathcal{K}_t^{\mathrm{FAILURE}}|\rfloor\big), \\ \mathcal{R}_t^{\mathrm{FAILURE}} &= \{k \in \mathcal{A}_t^{\mathrm{FAILURE}} : \mathrm{rank}_{\mathrm{fail}}(k) \leq q_t^{\mathrm{FAILURE}}\}, \\ b_t &= b_{t-1} + \beta_{\mathrm{FAILURE}}|\mathcal{K}_t^{\mathrm{FAILURE}}| - q_t^{\mathrm{FAILURE}}. \end{aligned} \tag{23}$$

The candidate count corresponds to the implementation's `lane_decidable`: it excludes initial creations, solved keys, and unusable servings, but includes failure keys subsequently rejected by admission. The quota is spent only on admitted keys. All unused allowance, including whole calls when too few keys are admitted, carries forward. The selected keys refresh now; unselected keys reuse their skills and must qualify again at a later serving.

This accounting preserves fractional allocations. For example, with $\beta_{\mathrm{FAILURE}} = 0.5$ and one admitted failure candidate per step, the first step carries 0.5 calls and the second can select one refresh. Thus β_{FAILURE} controls cumulative allocation rather than a fixed retained fraction at each step:

$$\sum_{u=1}^{t} |\mathcal{R}_u^{\mathrm{FAILURE}}| = \beta_{\mathrm{FAILURE}} \sum_{u=1}^{t} |\mathcal{K}_u^{\mathrm{FAILURE}}| - b_t \leq \beta_{\mathrm{FAILURE}} \sum_{u=1}^{t} |\mathcal{K}_u^{\mathrm{FAILURE}}|. \tag{24}$$

This bound accounts for controller-selected failure refreshes. Initial creations, contrast refreshes, and extraction fallbacks are separate analyzer costs. Prompt lengths differ, so it is not a token budget. At $\beta_{\mathrm{FAILURE}} = 1$, the allowance covers every admitted failure key, and ranking does not affect selection. Smaller rates make ranking determine the selected subset when the quota is insufficient.

Contrast Threshold Filtering The threshold $\tau_{\mathcal{U}}$ filters contrast keys by requiring sufficient remaining failure mass, $E_k(t) > \tau_{\mathcal{U}}$. Raising it makes admission more restrictive. Together with the mixed-outcome and persistence conditions, this determines the contrast refresh set: all admitted contrast keys refresh in the reported experiments, so

$$\begin{aligned} \mathcal{R}_t^{\mathrm{CONTRAST}} = \mathcal{A}_t^{\mathrm{CONTRAST}} \subseteq \{k \in \mathcal{K}_t^{\mathrm{CONTRAST}} : E_k(t) > \tau_{\mathcal{U}}\}, \\ \mathcal{R}_t = \mathcal{R}_t^{\mathrm{FAILURE}} \cup \mathcal{R}_t^{\mathrm{CONTRAST}}. \end{aligned} \tag{25}$$

At fixed rollout count m, increasing $\tau_{\mathcal{U}}$ removes keys with larger success counts. With different sample sizes, the Wilson statistic also weights the strength of the evidence: a key at $5/32$ has higher E_k than one at $1/8$, despite its higher empirical success rate. Raising the threshold therefore filters on evidence-weighted remaining failure rather than a universal empirical success-rate cutoff. For fixed rollout evidence and history, a higher threshold can only remove admitted keys. The fraction removed depends on the observed scores; it is not a fixed sampling ratio.

Defaults and Ablations Our main results uses $\beta_{\mathrm{FAILURE}} = 1$ and $\tau_{\mathcal{U}} = 0.05$, selecting almost every admitted key in both lanes. Its reduction in refresh calls comes from the admission rules. TI-BF changes β_{FAILURE} to 0.5, reducing the failure quota accrual rate while retaining the same admission rule. TI-TC changes $\tau_{\mathcal{U}}$ to 0.2152, tightening contrast admission while leaving its quota configuration unchanged.

B.5 Adaptive Schedule: Derivations

This section specifies the RL and OPD objectives used in Section 3.3, derives the training schedule and optimizer-dependent conflict bounds, and clarifies the assumptions and approximations applied to our implementation.

B.5.1 RL and Distillation Objectives

Training Objective Following the outcome-based RL and privileged-context distillation formulation of SEED (Wu et al., 2026), let τ denote an interaction trajectory for task x. At turn u, the policy context $c_{\tau,u} = (x, o_0, a_0, \ldots, o_u)$ contains the task and observable interaction history, where o_u is the environment observation and $a_u \sim \pi_\theta(\cdot \mid c_{\tau,u})$ is the action sampled from policy π_θ. The completed trajectory receives an outcome reward $R(\tau)$, and the RL objective is to maximize the expected reward $J(\theta) = \mathbb{E}_{x,\tau\sim\pi_\theta(\cdot|x)}[R(\tau)]$. Skills are only applied to the teacher's context. Our external analyzer and persistent skill bank supply this privileged context, as described in Section 3.1.

At training step t, the behavior snapshot $\pi_{\theta_{\mathrm{old}}}$ collects the trajectories. The actor combines reward-driven policy learning with skill-conditioned teacher guidance by minimizing

$$\mathcal{L}(\theta; t) = \mathcal{L}_{\mathrm{RL}}(\theta) + \lambda_t \mathcal{L}_{\mathrm{OPD}}(\theta), \qquad \mathcal{L}_{\mathrm{RL}} = \mathcal{L}_{\mathrm{PG}} + \kappa_{\mathrm{KL}}\mathcal{L}_{\mathrm{KL}} - \kappa_{\mathrm{H}}\mathcal{H}. \tag{26}$$

Here, $\mathcal{L}_{\mathrm{PG}}$ is the PPO policy-gradient surrogate: it uses outcome-based, group-relative advantages to favor higher-reward actions, with importance-ratio clipping relative to $\pi_{\theta_{\mathrm{old}}}$. $\mathcal{L}_{\mathrm{KL}}$ is a sampled-token KL penalty that regularizes deviation from a fixed reference policy π_{ref}, initialized from the actor's starting checkpoint and held fixed throughout training. $\mathcal{H}$ is the current policy's token-distribution entropy; its negative sign encourages exploration by rewarding higher entropy. The fixed, nonnegative coefficients κ_{KL} and κ_{H} set these regularization strengths.

The masked $\mathcal{L}_{\mathrm{OPD}}$ provides skill-conditioned guidance through sigmoid-weighted reinforcement of sampled tokens. Teacher scores and gate weights are detached. Stage 2 in Section 3.3 controls this guidance through λ_t, applying a bounded reduction when its estimated conflict with the RL signal exceeds the tolerance. The selected λ_t remains fixed within each training step.

Rows, Masks, and Advantages Each turn row r has sampled tokens $y_{r,i}$, context c_r, and response mask $m_{r,i} \in \{0, 1\}$. Define the masked mean, token log-probability, and importance ratio as

$$\begin{aligned} \langle z \rangle_m &:= \frac{1}{Z_m} \sum_{r,i} m_{r,i} z_{r,i}, \qquad Z_m = \sum_{r,i} m_{r,i} + \epsilon, \\ \ell_{r,i}(\theta) &= \log \pi_\theta(y_{r,i} \mid c_r, y_{r,<i}), \qquad \rho_{r,i}(\theta) = \exp\big(\ell_{r,i}(\theta) - \ell_{r,i}^{\text{old}}\big). \end{aligned} \tag{27}$$

Here $\epsilon = 10^{-8}$ stabilizes the masked mean, and $\ell_{r,i}^{\text{old}} = \log \pi_{\theta_{\text{old}}}(y_{r,i} \mid c_r, y_{r,<i})$ denotes the behavior-snapshot log-probability. Row rewards R_r, including a 0.1 invalid-action penalty, are normalized over all turn rows in each prompt group g and broadcast as token advantages. ALFWorld and SearchQA use $A_r^{\text{out}} = (R_r - \mu_g)/(\sigma_g + \varepsilon)$: they subtract the group mean reward μ_g and divide by the sample standard deviation σ_g, with $\varepsilon = 10^{-6}$. WebShop uses $A_r^{\text{out}} = R_r - \mu_g$, subtracting the same group mean without standard-deviation scaling. Equation 11 defines the outcome-based advantages.

Clipped Policy Surrogate The default `vanilla` loss in the launch configurations uses token-level dual-clipped PPO. For detached advantage $A_{r,i}$, let

$$\begin{aligned} C_{r,i}(\theta) &= \max\{-A_{r,i}\rho_{r,i},\ -A_{r,i}\,\text{clip}(\rho_{r,i}, 1-\epsilon_-, 1+\epsilon_+)\}\,, \\ \ell_{r,i}^{\text{PG}}(\theta) &= \begin{cases} \min\{C_{r,i}(\theta), -c_{\text{dual}}A_{r,i}\}, & A_{r,i} < 0, \\ C_{r,i}(\theta), & A_{r,i} \geq 0, \end{cases} \qquad \mathcal{L}_{\text{PG}} = \langle \ell^{\text{PG}} \rangle_m. \end{aligned} \tag{28}$$

Here $\epsilon_- = \epsilon_+ = 0.2$ and $c_{\text{dual}} = 3$ in the default configuration. The second clip limits the loss contribution of negative-advantage tokens at large importance ratios. The reduction is a masked token mean within each microbatch, followed by the trainer's gradient accumulation. The implementation also supports GSPO, which instead uses sequence-level ratios and a mean of per-row token means; it is a separate configurable surrogate.

KL and Entropy Terms Let $\ell_{r,i}^{\text{ref}} = \log \pi_{\text{ref}}(y_{r,i} \mid c_r, y_{r,<i})$ be the fixed reference policy's log-probability and $d_{r,i} = \ell_{r,i}^{\text{ref}} - \ell_{r,i}(\theta)$. With a fixed per-token clipping bound $C_{\text{KL}} = 10$, the implemented regularizers are

$$\begin{aligned} \mathcal{L}_{\text{KL}} &= \big\langle \text{clip}\big(e^d - d - 1, -C_{\text{KL}}, C_{\text{KL}}\big) \big\rangle_m\,, \\ \mathcal{H} &= \left\langle -\sum_a \pi_\theta(a \mid c_r, y_{r,<i}) \log \pi_\theta(a \mid c_r, y_{r,<i}) \right\rangle_m. \end{aligned} \tag{29}$$

The KL expression is the sampled-token `low_var_kl` estimator, with extreme penalty values clipped before masked averaging. The bound C_{KL} is a numerical safeguard with value 10. Benchmark-specific coefficients are listed in Appendix D.2.

Gated OPD Loss The skill-conditioned teacher scores the sampled tokens before actor optimization and caches $\ell_{r,i}^{\text{teach}}$ for the update. With supervision mask $M_{r,i} \leq m_{r,i}$ and stop-gradient operator sg, define

$$\begin{aligned} \omega_{r,i} &= \text{sg}\big[\text{sigmoid}\big(\beta_{\text{OPD}}[\ell_{r,i}^{\text{teach}} - \ell_{r,i}(\theta)]\big)\big]\,, \\ \mathcal{L}_{\text{OPD}} &= \big\langle \omega\,[\text{sg}(\ell^{\text{teach}}) - \ell(\theta)] \big\rangle_M\,, \qquad G_D = -\frac{1}{Z_M} \sum_{r,i} M_{r,i}\omega_{r,i}\nabla_\theta \ell_{r,i}(\theta). \end{aligned} \tag{30}$$

Following SEED (Wu et al., 2026), we use $\beta_{\text{OPD}} = 5$. The gate is recomputed at each loss evaluation but detached, yielding a weighted negative-log-likelihood gradient G_D. This sampled loss is not a full-distribution KL. Below, $G_R = \nabla_\theta \mathcal{L}_{\text{RL}}$ and G_D denote parameter-space gradients; Appendix B.5.5 distinguishes the controller's proxies.

B.5.2 Stage 1: Progress Estimation and Clock Bounds

Success-rate Noise Scale Let h be a positive integer, $H = 2h$, and $T > 0$. Define the indexed collection of historical differences

$$D_t = (\text{SR}_i - \text{SR}_{i-1})_{i=2}^{t-H}, \tag{31}$$

taking it to be empty when $t - H < 2$. The noise scale in Equation 6 uses the convention

$$\varsigma_t = \begin{cases} 0, & |D_t| < 2, \\ \sqrt{2/h}\ \mathrm{std}_{\mathrm{pop}}(D_t), & |D_t| \geq 2, \end{cases} \tag{32}$$

where $|D_t|$ counts the historical differences and $\mathrm{std}_{\mathrm{pop}}$ is their population standard deviation. The scale ς_t governs the next clock increment $v_{t+1} - v_t$.

Equation 6 measures progress from the most recent H success-rate observations and estimates noise from changes between consecutive observations before that window. With $H = 10$ by default, the implementation accumulates these earlier differences' count, sum, and squared sum online. The clock advances by one per step during warm-up. Even after the progress window fills, the clock advances by one if fewer than two historical differences are available or their standard deviation is zero. Initialization at $v_1 = 0$ gives $\lambda_1^{\mathrm{raw}} = \lambda_{\max}$.

We use first differences so that steady linear improvement does not inflate the noise estimate: a linear trend becomes a constant offset in the differences, which their standard deviation removes. The factor $\sqrt{2/h}$ corresponds to comparing two means of h observations. Under a deterministic linear trend plus independent observation noise of variance σ^2, each half-window mean contributes variance σ^2/h, so

$$\mathrm{Var}(\Delta_t) = \frac{\sigma^2}{h} + \frac{\sigma^2}{h} = \frac{2\sigma^2}{h}, \qquad \mathrm{sd}(\Delta_t) = \sqrt{\frac{2}{h}}\,\sigma. \tag{33}$$

This motivates the multiplier for an observation-noise estimate. The implementation instead multiplies the standard deviation of historical *first differences*, whose population variance is $2\sigma^2$ under this model. Its population scale is therefore $\varsigma_{\mathrm{pop}} = \sqrt{2/h}\,\sqrt{2}\sigma = \sqrt{2}\,\mathrm{sd}(\Delta_t)$. The chosen normalization retains the $h^{-1/2}$ dependence but tempers the normalized progress signal through this larger population scale. It is **a heuristic normalization, not an exact standard-error estimate**.

Using observations before the progress window makes the noise estimate independent of the current progress statistic under independent observation noise, but not necessarily for autocorrelated training data. The clock result below requires the stated sign symmetry; it neither requires exact scale calibration nor uniquely determines the multiplier.

Proposition 1 (Unit expected clock advance)**.** *Let $\phi_{a,b}(u) = \mathrm{clip}(1 + u, a, b)$ for finite $a \leq 1 \leq b$. Then*

$$\phi_{a,b}(u) + \phi_{a,b}(-u) = 2 \quad \textit{for all } u \quad \Longleftrightarrow \quad a + b = 2. \tag{34}$$

For finite Δ_t and $\varsigma_t \geq 0$, define $u_t = \Delta_t/\varsigma_t$ when $\varsigma_t > 0$ and $u_t = 0$ otherwise. If

$$(\Delta_t, \varsigma_t) \overset{d}{=} (-\Delta_t, \varsigma_t), \tag{35}$$

then $\mathbb{E}[\phi_{a,b}(u_t)] = 1$ whenever $a + b = 2$. In particular, the implemented bounds $[0, 2]$ give $\mathbb{E}[v_{t+1} - v_t] = 1$.

For the scale in Equation 6, a sufficient condition for Equation 35 is that the two half-window means are exchangeable conditional on the observations preceding the window. This condition holds for i.i.d. success-rate observations from a fixed distribution; their individual distribution need not be symmetric.

Proof. Clipping symmetry. If $a + b = 2$, set $c = b - 1 = 1 - a \geq 0$. Then

$$\phi_{a,b}(u) = 1 + \mathrm{clip}(u, -c, c), \qquad \mathrm{clip}(-u, -c, c) = -\,\mathrm{clip}(u, -c, c). \tag{36}$$

The second identity follows by checking $u < -c$, $-c \leq u \leq c$, and $u > c$: the clipped values are respectively $-c$, u, and c, with opposite values at $-u$. Adding the two expressions for ϕ gives Equation 34. Conversely, choose $u \geq \max\{b - 1, 1 - a\}$. Then $\phi_{a,b}(u) = b$ and $\phi_{a,b}(-u) = a$, so the identity for all u requires $a + b = 2$.

Expected advance. The map $f(d, s) = d/s$ for $s > 0$, with $f(d, 0) = 0$, satisfies $f(-d, s) = -f(d, s)$, including at $s = 0$. Applying this map to Equation 35 gives $u_t \overset{d}{=} -u_t$. Since $\phi_{a,b}$ is bounded, its expectation exists, and

$$\begin{aligned} 2\mathbb{E}[\phi_{a,b}(u_t)] &= \mathbb{E}[\phi_{a,b}(u_t)] + \mathbb{E}[\phi_{a,b}(-u_t)] \\ &= \mathbb{E}[\phi_{a,b}(u_t) + \phi_{a,b}(-u_t)] = 2. \end{aligned} \tag{37}$$

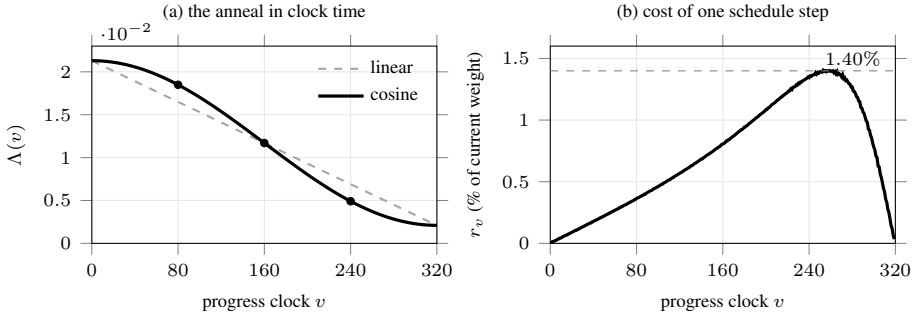


Figure 7: Cosine annealing illustrated with $[\lambda_{\min}, \lambda_{\max}] = [0.0021, 0.0213]$ and $T = 320$. **(a)** Weight versus progress-clock position, with a linear anneal for comparison. **(b)** Relative one-step reduction $r(v) = 1 - \Lambda(v+1)/\Lambda(v)$, which bounds Stage 2's correction and peaks at approximately 1.40% for this λ_t range and schedule horizon.

For $[a, b] = [0, 2]$, Equation 6 sets $v_{t+1} - v_t = \phi_{0,2}(u_t)$, proving the claim.

A sufficient condition for signal symmetry. For $t \geq H$, let X_t and Y_t be the recent and preceding half-window means, so $\Delta_t = X_t - Y_t$. Let $\mathcal{F}_{t-H} = \sigma(\mathrm{SR}_1, \ldots, \mathrm{SR}_{t-H})$ denote the earlier observation history. All differences used in ς_t end at or before $t - H$, so the scale is determined by $\mathcal{F}_{t-H}$. Conditional exchangeability means

$$\mathcal{L}\big((X_t, Y_t) \mid \mathcal{F}_{t-H}\big) = \mathcal{L}\big((Y_t, X_t) \mid \mathcal{F}_{t-H}\big), \tag{38}$$

where $\mathcal{L}(\cdot \mid \mathcal{F})$ denotes conditional distribution. Swapping X_t, Y_t negates Δ_t and leaves the history-dependent scale unchanged. Therefore,

$$\mathcal{L}\big((\Delta_t, \varsigma_t) \mid \mathcal{F}_{t-H}\big) = \mathcal{L}\big((-\Delta_t, \varsigma_t) \mid \mathcal{F}_{t-H}\big). \tag{39}$$

Taking expectations over the earlier history proves Equation 35. Under i.i.d. observations, the two disjoint blocks each contain h samples from the same distribution and are independent of the earlier history. Their means are consequently conditionally exchangeable, establishing the sufficient condition. During warm-up, $\Delta_t = 0$; whenever the scale is zero, $u_t = 0$. Both cases give unit advance directly. □

With nonnegative increments and a possible stall, symmetry fixes the bounds to $[0, 2]$. Under the stated **signal-symmetry assumption** at every step, $\mathbb{E}[v_t] = t - 1$, even with dependent increments. This guarantees unit expected **clock advance** but does not guarantee equal cumulative distillation weight, because Λ is nonlinear.

Cosine Shape and Clamping For $0 < v < T$, differentiating Equation 7 gives

$$\Lambda'(v) = -\frac{\pi(\lambda_{\max} - \lambda_{\min})}{2T} \sin\frac{\pi v}{T}, \qquad \Lambda''(v) = -\frac{\pi^2(\lambda_{\max} - \lambda_{\min})}{2T^2} \cos\frac{\pi v}{T}. \tag{40}$$

The endpoint slopes vanish and the maximum decay rate occurs at $T/2$. The curve is concave in its first half and convex in its second. The clamp fixes $\Lambda(v) = \lambda_{\min}$ for $v \geq T$, including when the progress clock reaches the horizon before training ends. Figure 7 shows the curve and the relative one-step reduction used by Stage 2.

B.5.3 Stage 2: First-order Conflict Conditions

Scope and Assumptions We first derive a first-order conflict condition for a single optimizer update, then specialize its history contribution to AdamW, and finally construct the estimates used by the training-step controller. Throughout this derivation, j indexes optimizer updates and t indexes training steps, each of which can contain several optimizer updates. We use h_j^{opt} for the exact normalized history contribution at update j and reserve h_t for its practical estimate computed once per training step.

For a fixed update point θ, fix the sampled batch, its advantages, and the reference and teacher scores. Let $G_R = \nabla_\theta \mathcal{L}_{\mathrm{RL}}$ and let G_D be the OPD backward gradient in Equation 30.

Assume $G_R \neq 0$, $\lambda \geq 0$, and that $\mathcal{L}_{\mathrm{RL}}$ has an L_R-Lipschitz gradient in a smooth neighborhood containing the proposed step. The bounds characterize the sampled RL loss within smooth regions of the clipped objectives; task return and validation performance are assessed empirically.

Lemma 1 (First-order undone fraction)**.** *For $\theta^+ = \theta - \eta(G_R + \lambda G_D)$ with $\eta > 0$, define*

$$U = -\frac{\lambda \langle G_R, G_D \rangle}{\|G_R\|^2}. \tag{41}$$

Then

$$\mathcal{L}_{\mathrm{RL}}(\theta^+) - \mathcal{L}_{\mathrm{RL}}(\theta) = -\eta \|G_R\|^2 (1 - U) + O(\eta^2). \tag{42}$$

The first-order term is nonpositive exactly when $U \leq 1$, equivalently $U^+ \leq 1$.

Proof. Let $d = G_R + \lambda G_D$. Local Lipschitz continuity bounds the Taylor remainder:

$$\mathcal{L}_{\mathrm{RL}}(\theta - \eta d) - \mathcal{L}_{\mathrm{RL}}(\theta) = -\eta \langle G_R, d \rangle + \mathcal{E}_\eta, \qquad |\mathcal{E}_\eta| \leq \frac{L_R \eta^2}{2} \|d\|^2. \tag{43}$$

Expanding the directional derivative and substituting U gives

$$\langle G_R, d \rangle = \|G_R\|^2 + \lambda \langle G_R, G_D \rangle = \|G_R\|^2 (1 - U). \tag{44}$$

Since $\eta \|G_R\|^2 > 0$, the linear term is nonpositive if and only if $1 - U \geq 0$. Because $1 > 0$, this is also equivalent to $\max(U, 0) \leq 1$. Substitution in Equation 43 proves the expansion. □

Meaning of the Unit Bound When the gradients oppose, U is the fraction of the pure-RL first-order decrease canceled by distillation; when they align, $U < 0$. The boundary $U^+ = 1$ motivates the fixed baseline tolerance $q^* = 1$, which later caps the controller's step-specific tolerance q_t^*. It is a *first-order* boundary: $U = 1$ leaves the sign of the quadratic remainder undetermined. If $U < 1$, $d \neq 0$, and $L_R > 0$, a sufficient finite-step condition within the smooth neighborhood is

$$0 < \eta \leq \frac{2\|G_R\|^2 (1 - U)}{L_R \|d\|^2}. \tag{45}$$

The controller does not enforce this learning-rate bound, so the lemma alone does not guarantee a finite-step loss decrease.

Proposition 2 (Preconditioned update with history)**.** *At optimizer update j, for a fixed symmetric positive definite P_j, a fixed history contribution B_j, and $\eta > 0$, consider*

$$\begin{gathered} \Delta\theta_j = -\eta \big[P_j (G_R + \lambda G_D) + B_j \big], \\ U_{P_j} = \frac{\lambda \langle G_R, -P_j G_D \rangle}{\|G_R\|_{P_j}^2}, \qquad h_j^{\mathrm{opt}} = \frac{\langle G_R, B_j \rangle}{\|G_R\|_{P_j}^2}, \qquad \|x\|_{P_j}^2 = \langle x, P_j x \rangle. \end{gathered} \tag{46}$$

Under the preceding smoothness assumptions, the first-order RL loss change is nonpositive exactly when $U_{P_j} \leq 1 + h_j^{\mathrm{opt}}$.

Proof. Put $d_P = P_j(G_R + \lambda G_D) + B_j$. Taylor expansion gives $\mathcal{L}_{\mathrm{RL}}(\theta + \Delta\theta_j) - \mathcal{L}_{\mathrm{RL}}(\theta) = -\eta \langle G_R, d_P \rangle + O(\eta^2)$, with remainder bounded by $L_R \eta^2 \|d_P\|^2 / 2$. Expanding all three contributions,

$$\begin{aligned} \langle G_R, d_P \rangle &= \langle G_R, P_j G_R \rangle + \lambda \langle G_R, P_j G_D \rangle + \langle G_R, B_j \rangle \\ &= \|G_R\|_{P_j}^2 (1 - U_{P_j} + h_j^{\mathrm{opt}}). \end{aligned} \tag{47}$$

Positive definiteness and $G_R \neq 0$ imply $\|G_R\|_{P_j}^2 > 0$. Dividing by this quantity preserves the inequality, giving $1 - U_{P_j} + h_j^{\mathrm{opt}} \geq 0$, or $U_{P_j} \leq 1 + h_j^{\mathrm{opt}}$. Setting $P_j = I$ and $B_j = 0$ recovers Lemma 1. □

Both U_{P_j} and h_j^{opt} are normalized by the same preconditioned RL contribution $\|G_R\|_{P_j}^2$. The condition therefore applies to conflict measured in this P_j metric. If $1 + h_j^{\mathrm{opt}} \geq 0$, it is equivalent to $U_{P_j}^+ \leq 1 + h_j^{\mathrm{opt}}$. If $1 + h_j^{\mathrm{opt}} < 0$, sufficiently aligned distillation is required to offset adverse history; attenuating an opposing term cannot restore first-order descent. We next identify P_j and B_j for AdamW, before introducing the controller's approximations.

B.5.4 AdamW History Contribution

Moment Recursions and Bias Correction At optimizer update j, initially take the input to be $g_j = G_R + \lambda G_D$, without gradient clipping. AdamW maintains raw first and second moments

$$\begin{aligned} m_j &= \beta_1 m_{j-1} + (1-\beta_1) g_j, & v_j &= \beta_2 v_{j-1} + (1-\beta_2) g_j \odot g_j, \\ \hat{m}_j &= \frac{m_j}{1-\beta_1^j}, & \hat{v}_j &= \frac{v_j}{1-\beta_2^j}, \qquad P_j = \operatorname{diag}\left(\frac{1}{\sqrt{\hat{v}_j}+\epsilon}\right). \end{aligned} \tag{48}$$

Here $\odot$ denotes elementwise multiplication, $0 \le \beta_1, \beta_2 < 1$, and $\epsilon > 0$. The moment vector v_j is unrelated to the scalar progress clock v_t. With decoupled weight-decay coefficient wd, the update is

$$\theta_{j+1} = (1 - \eta \,\mathrm{wd})\theta_j - \eta P_j \hat{m}_j, \qquad \Delta\theta_j = -\eta \left[\frac{P_j m_j}{1-\beta_1^j} + \mathrm{wd}\,\theta_j\right]. \tag{49}$$

Separating Current Gradient from History Substituting the first-moment recursion into Equation 49 and factoring the positive scalar $a_j = (1-\beta_1)/(1-\beta_1^j)$ gives

$$\begin{aligned} \Delta\theta_j &= -\eta \left[\frac{1-\beta_1}{1-\beta_1^j} P_j g_j + \frac{\beta_1}{1-\beta_1^j} P_j m_{j-1} + \mathrm{wd}\,\theta_j\right] \\ &= -\eta a_j \left[P_j g_j + \underbrace{\frac{\beta_1}{1-\beta_1} P_j m_{j-1}}_{B_j^{\mathrm{mom}}} + \underbrace{\frac{1-\beta_1^j}{1-\beta_1}\,\mathrm{wd}\,\theta_j}_{B_j^{\mathrm{wd}}}\right]. \end{aligned} \tag{50}$$

Absorbing $a_j > 0$ into the learning rate leaves the first-order sign unchanged. At the realized update, Proposition 2 therefore applies with $B_j = B_j^{\mathrm{mom}} + B_j^{\mathrm{wd}}$ and gives

$$\begin{aligned} h_j^{\mathrm{opt}} &= \frac{\langle G_R, B_j^{\mathrm{mom}} + B_j^{\mathrm{wd}} \rangle}{\|G_R\|_{P_j}^2} \\ &= \underbrace{\frac{\beta_1}{1-\beta_1} \frac{\langle G_R, P_j m_{j-1} \rangle}{\|G_R\|_{P_j}^2}}_{\text{momentum contribution}} + \underbrace{\frac{1-\beta_1^j}{1-\beta_1}\,\mathrm{wd}\,\frac{\langle G_R, \theta_j \rangle}{\|G_R\|_{P_j}^2}}_{\text{weight-decay contribution}}. \end{aligned} \tag{51}$$

This is the same h_j^{opt} defined in Proposition 2, with AdamW's history terms substituted. Its denominator uses the same P_j as the conflict U_{P_j}, so the exact first-order condition remains $U_{P_j} \le 1 + h_j^{\mathrm{opt}}$. Positive h_j^{opt} supports RL descent and increases the allowable opposing distillation; negative h_j^{opt} reduces that allowance. The controller estimates this quantity before the update, as described in Section B.5.5.

Approximation When Selecting λ Equations 49–51 describe the realized, unclipped AdamW update. However, its current second moment, and hence P_j and B_j^{mom}, depend on the combined gradient $G_R + \lambda G_D$. To select λ beforehand, the controller substitutes a preconditioner computed from the previous second moment. This gives an available estimate of the history contribution; it is not the exact contribution for the eventual update.

Effect of Gradient Clipping The actor clips the combined gradient before calling AdamW. If this multiplies the input by $c_j \in (0, 1]$, then the exact decomposition at that update has responding term $c_j P_j (G_R + \lambda G_D)$ and the same history terms in Equation 50, with P_j evaluated from the actual clipped input. Thus

$$P_j^{\mathrm{eff}} = c_j P_j, \qquad U_{P_j^{\mathrm{eff}}} = U_{P_j}, \qquad h_j^{\mathrm{opt,clip}} = \frac{h_j^{\mathrm{opt}}}{c_j}. \tag{52}$$

At the actual preconditioner P_j, the clipping factor cancels between the numerator and denominator of U_{P_j}. However, clipping reduces the current-gradient contribution without directly scaling momentum

or weight decay, so their relative contribution becomes h_j^{opt}/c_j. Thus, clipping strengthens the effect of history on the tolerance: helpful history increases the allowance further, while adverse history reduces it further. **The controller estimates this tolerance before the update, without the eventual clipping factor or current preconditioner**, both of which depend on the selected λ. Its estimate therefore **approximates the tolerance for the applied AdamW update**.

B.5.5 Online Estimation and Limits of the Derivation

The exact condition concerns one optimizer update and its actual parameter-gradient geometry. The controller instead selects one λ_t before the optimizer updates in training step t. It therefore estimates the history contribution and the conflict separately, then uses them to set a bounded correction.

History Measurement Let $j(t)$ be the first optimizer-update index in training step t. Before that update, one additional forward/backward pass on the first microbatch gives its policy-surrogate parameter gradient $\widetilde{G}_{R,t}$. When complete optimizer state is available and $j(t) > 1$, define

$$\widetilde{P}_t = \mathrm{diag}\left(\frac{1}{\sqrt{\hat{v}_{j(t)-1}} + \epsilon}\right), \qquad \hat{v}_{j(t)-1} = \frac{v_{j(t)-1}}{1-\beta_2^{j(t)-1}}. \tag{53}$$

For a positive denominator, substituting $\widetilde{G}_{R,t}$ and $\widetilde{P}_t$ into Equation 51 defines the training-step estimate

$$\begin{aligned} h_t = {} & \frac{\beta_1}{1-\beta_1}\frac{\langle \widetilde{G}_{R,t}, \widetilde{P}_t m_{j(t)-1}\rangle}{\|\widetilde{G}_{R,t}\|^2_{\widetilde{P}_t}} \\ & + \frac{1-\beta_1^{j(t)}}{1-\beta_1}\,\mathrm{wd}\,\frac{\langle \widetilde{G}_{R,t}, \theta_{j(t)}\rangle}{\|\widetilde{G}_{R,t}\|^2_{\widetilde{P}_t}}. \end{aligned} \tag{54}$$

Scalar products are reduced across workers. The raw momentum and second moment come from completed updates, while the weight-decay factor uses the upcoming index $j(t)$. We reuse h_t throughout training step t; it is an estimate for the upcoming update, not an average over that step's optimizer updates. Missing or incomplete optimizer state, or an invalid denominator, triggers the unit-tolerance fallback below.

Thus h_t approximates $h_{j(t)}^{\mathrm{opt}}$ using one microbatch, the previous preconditioner, and the policy-surrogate gradient. It excludes KL, entropy, and the future gradient-clipping factor, and need not represent the history contribution at later optimizer updates in the same training step.

Log-probability Conflict Proxy The implementation evaluates $q_t = \lambda_t^{\mathrm{raw}}(-c_t)^+ N_D/N_R$ at update start, where $N_R = \|g_R\|$, $N_D = \|g_D\|$, and $c_t = \langle g_R, g_D\rangle/(N_R N_D)$. For token-mean reduction and inactive PPO clipping at $\theta = \theta_{\mathrm{old}}$, the policy-surrogate and OPD derivatives with respect to token log-probabilities are

$$(g_R)_{r,i} = -\frac{m_{r,i}A_{r,i}}{Z_m}, \qquad (g_D)_{r,i} = -\frac{M_{r,i}\omega_{r,i}}{Z_M}. \tag{55}$$

These quantities use the actual masks and loss-reduction weights and require no additional parameter backward pass. They describe the policy-surrogate component, however, rather than the full regularized $\mathcal{L}_{\mathrm{RL}}$ in Equation 26.

To relate this proxy to the proposition's P_j metric, let $\mathcal{J} = \partial \ell/\partial\theta$ be the token-log-probability Jacobian. For the policy-surrogate component,

$$\begin{aligned} G_{\mathrm{PG}} = \mathcal{J}^\top g_R, \qquad G_D = \mathcal{J}^\top g_D, \\ \langle G_{\mathrm{PG}}, P_j G_D\rangle = g_R^\top \mathcal{J} P_j \mathcal{J}^\top g_D, \qquad \|G_{\mathrm{PG}}\|^2_{P_j} = g_R^\top \mathcal{J} P_j \mathcal{J}^\top g_R. \end{aligned} \tag{56}$$

The measured q_t approximates the positive conflict $U^+_{P_j}$ at $\lambda = \lambda_t^{\mathrm{raw}}$ by replacing $\mathcal{J}P_j\mathcal{J}^\top$ with a positive scalar multiple of the identity; that scalar cancels in the ratio. This is a separate approximation from estimating h_t: the controller does not evaluate conflict in the exact parameter-space metric used by the proposition. The proxy inner product need not preserve the sign of the parameter-space inner product, and the proxy omits KL and entropy gradients. Thus combining q_t with h_t does not inherit the exact first-order guarantee. We evaluate the controller's practical effectiveness through the Stage 2 ablation in Section 4.3.

Adaptive Tolerance For a nonnegative history-adjusted allowance, the condition $U_{P_j}^{+} \le 1 + h_j^{\mathrm{opt}}$ motivates comparing the proxy q_t with $1+h_t$. Equation 10 gives the nominal tolerance $\min(q^*, 1+h_t)$, with $q^* = 1$. In implementation, an available, finite history estimate instead gives the applied tolerance $q_t^* = \min\{1, \max(\epsilon_q, 1 + h_t)\}$, where $\epsilon_q = 10^{-12}$. Helpful estimated history therefore leaves the tolerance at 1, while adverse estimated history lowers it to no less than ϵ_q. This floor keeps the correction defined even when the estimated allowance is nonpositive; it does not restore the proposition's condition in that case. A missing or nonfinite h_t falls back to $q_t^* = 1$; fixed-tolerance variants also use 1. In the implementation details below, q_t^* denotes this applied tolerance.

For fixed measured signals and $q_t > 0$, conflict scales linearly with λ_t. Hence $\lambda_t^{\mathrm{raw}} \min(1, q_t^*/q_t)$ is the largest weight no greater than the base weight that meets the measured tolerance. Applying the additional schedule floor gives Equation 8 and can prevent that tolerance from being reached.

Bounded Correction For $\lambda_{\min} > 0$, Equation 8 gives

$$0 \le \frac{\lambda_t^{\mathrm{raw}} - \lambda_t}{\lambda_t^{\mathrm{raw}}} \le r(v_t) := 1 - \frac{\Lambda(v_t + 1)}{\Lambda(v_t)}, \qquad q_t^{\mathrm{applied}} = q_t \frac{\lambda_t}{\lambda_t^{\mathrm{raw}}}. \tag{57}$$

For $q_t > q_t^*$, the measured tolerance is attainable exactly when $q_t[1 - r(v_t)] \le q_t^*$. Otherwise the correction reaches its floor with residual conflict. For any realized clock path,

$$0 \le \sum_{t=1}^{T} (\lambda_t^{\mathrm{raw}} - \lambda_t) \le \sum_{t=1}^{T} \big[\Lambda(v_t) - \Lambda(v_t + 1)\big]. \tag{58}$$

This compares weights on the same clock path. Corrections do not recursively reduce future base weights, although they may affect later learning and hence later success rates. The bound depends on the λ_t range $[\lambda_{\min}, \lambda_{\max}]$ and the schedule horizon T; it is not a universal percentage or a guarantee of non-increasing RL loss.

Timing and Boundary Cases Stage 1 reads $\Lambda(v_t)$ before staging the clock advance from the current success rate. Stage 2 follows skill selection and teacher scoring, applies one coefficient throughout the training step, and commits the clock only after a successful update. **If either gradient proxy has zero norm, the correction retains the base weight and the conflict ratio is unavailable; measured $q_t = 0$ also retains the base weight**. At $v_t \ge T$, both schedule bounds equal $\lambda_{\min}$. If the history estimate is unavailable, the controller uses the default tolerance $q_t^* = 1$. If the conflict inputs or computed statistics are numerically invalid, it prevents the actor update and leaves the committed controller state unchanged.

The unconditional property of the refinement law is its **bounded coefficient change**: Equation 57 holds for any valid proxy and positive applied tolerance. It does not imply non-increasing RL loss, improved expected return, or complete removal of gradient conflict. The learning-rate and smoothness conditions of the proofs, the optimizer-state approximations, and the binding schedule floor prevent those stronger conclusions.

C Framework Implementation

This section describes how **TITRATE** is implemented on Verl (Sheng et al., 2025), with a focus on where each of the three components of Section 3 enters one training step and how the resulting state is kept consistent across the policy, the skill bank and the controllers (Figure 8). The implementation is organised as two packages that Verl's training loop calls at a small number of fixed points: a `seed` package holding the **versioned skill bank**, the grouping and serving logic and the analyzer client (Section C.2), and a `controllers` package holding the REUSE/REFRESH controller (Component B) and the two-stage adaptive schedule (Component A) behind one transactional facade (Section C.3). Extension points are collected in Section C.4.

C.1 One Training Step

One training step includes rollout, skill serving, teacher scoring, and actor optimization, which may include multiple optimizer updates. TITRATE connects these operations through the following sequence (Figure 8).

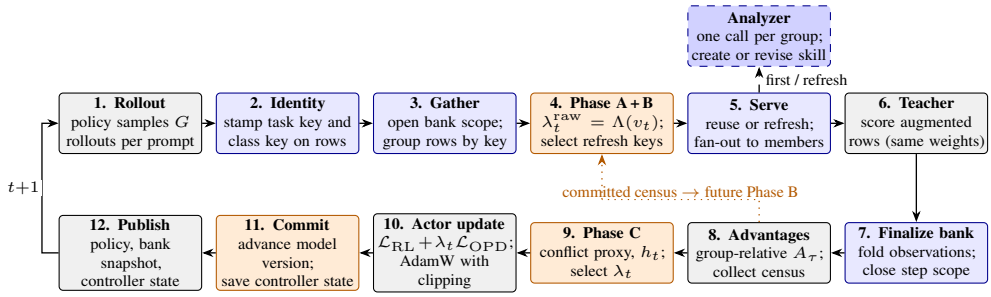


Figure 8: **One TITRATE training step inside Verl's training loop.** Grey boxes show rollout, scoring, optimization, and checkpoint publication; blue boxes show skill-bank operations; orange boxes show controller operations. Phase B selects refreshes from current rollout outcomes and prior serving history before teacher scoring. Bank observations are finalized before advantages and the new census are computed. Phase C then selects λ_t using the current conflict proxy and optional history estimate h_t. Controller state is committed after a successful actor update, and checkpoints save policy, bank, and controller state together.

1. **Rollout.** The policy samples G trajectories per prompt through the environment interaction loop.
2. **Task identity.** Trajectory rows carry task or class identities, allowing skill-bank entries to be identified across training steps (Section 3.1).
3. **Grouping.** The bank enters a step scope and groups eligible trajectories by the configured task or class key. Outcome-stratified selection determines the trajectory examples supplied to the analyzer.
4. **Base weight and refresh decision.** Phase A reads $\lambda_t^{\text{raw}} = \Lambda(v_t)$ and determines the next clock value using the current success rate. Phase B selects refreshes before serving: current rollout outcomes supply success counts, while advantage-flat statistics come from each key's last completed serving (Sections 3.3 and 3.2).
5. **Skill serving.** First encounters and admitted refreshes invoke the analyzer; reuse retrieves the stored skill. The resulting skill is attached to the group's trajectories (Appendix B.2).
6. **Teacher scoring.** The actor scores the original responses under skill-augmented contexts. When asynchronous execution is enabled, analysis and teacher preparation execute in parallel with student, reference, and value computations.
7. **Bank finalization.** Once teacher and pre-update student scores are available, the trainer folds value and exposure observations into the bank and closes its step scope.
8. **Advantages and census.** The trainer computes group-relative advantages (Equation 11) and collects the completed serving census, including advantage-flat statistics, for the controller's later commit.
9. **Conflict correction.** Phase C computes the log-probability conflict proxy and, when enabled, obtains h_t from the policy-surrogate probe (Appendix B.5.5). It selects one λ_t for all microbatches and inner epochs. Numerically invalid conflict inputs prevent the actor update.
10. **Actor optimization.** The actor optimizes the joint RL and masked OPD objective (Equation 26) with AdamW and gradient clipping, using λ_t.
11. **Controller commit.** After the actor update returns successfully, the bank's model-version counter advances and the controller commits its clock, weight, cumulative distillation weight, and serving history. If controller-directed serving has occurred but the actor update fails, training stops so the next recovery run can restore from a consistent checkpoint.
12. **Checkpoint publication.** At checkpoint steps, policy artifacts, the matching bank snapshot, and controller state are saved before the checkpoint is marked complete.

Why this order matters Refresh selection precedes teacher scoring because the selected skill determines the teacher's conditioning. Conflict correction follows scoring and advantage computation because it needs the signals produced for the current update.

C.2 VERSIONED SKILL BANK SERVICE

Configuration The training YAML file configures the bank under `algorithm.seed.bank`: `key_granularity` selects task-level or class-level keys, and `class_depth` sets the class hierarchy depth (Appendix B.1). `policy` selects the reuse/refresh rule. `outcome_split` selects either one mixed or separate success/failure entries. `skill_evolution` enables revision of the skill in {incumbent} variable during refresh, and `control_fraction` sets the fraction of keys that retain their initial skill. All our experiments use `outcome_split=false`, `skill_evolution=true`, and `control_fraction=0`; key granularity and serving policy vary across experimental settings (Section D.2).

Resume compatibility The bank records the settings that determine skill identity and interpretation, including key granularity, class taxonomy, prompt version, outcome splitting, and revision mode. Loading a checkpoint checks these settings against its saved metadata, preventing a resumed run from mixing incompatible skills or statistics. A fresh run cannot adopt a directory containing prior bank state without a validated checkpoint restore.

Storage and provenance Each entry stores one skill, its version, birth statistics, and accumulated observations (Appendix B.3). Three output artifacts preserve this information: an append-only skill-version log, a stream of per-step statistics, and checkpoint snapshots. Together, these records support version tracking, offline analysis, and restoration alongside the policy and controller.

Analyzer service The analyzer is configured under `algorithm.seed`, with `analysis_model` selecting the model. The main experiments use GPT-4.1 (Section 4.1); additional analyzer comparisons use Claude Sonnet 4.6 (`claude-sonnet-4-6`), Gemini 2.5 Flash (`gemini-2.5-flash`), and GPT-4.1 mini (`gpt-4.1-mini`). Request concurrency, prompt length, and completion length are controlled by `analysis_num_workers`, `analysis_max_prompt_chars`, and `analysis_max_completion_tokens`, respectively. Calls record provenance, request attempts, and available provider-reported token usage, with missing usage identified separately. A failed refresh can retain the previous skill. Under the default failure guard, analysis errors that leave the step without any usable skill stop training; a "no common pattern" response is not treated as an analyzer failure.

C.3 CONTROLLER RUNTIME

Execution interface The trainer calls four methods: `begin_step` (Phase A), `decide_refresh` (Phase B), `resolve_lambda` (Phase C), and `end_step` (commit). The first three methods compute decisions and prepare updates to the controller's state. After a successful actor update, `end_step` adds the completed serving census, validates the record, and applies the pending updates. Shared controller functions support both live execution and offline replay.

Configuration The training YAML configures the runtime under `algorithm.controller`: `enable` activates it; `lambda_shadow` records proposed weights without applying them, while `drive_refresh` independently enables controller-directed refreshes with `algorithm.seed.bank.policy=controller`. The variables `lambda_min` and `lambda_max` set the λ_t range, `shape` selects cosine or linear annealing, and `d_anneal` sets the schedule horizon, defaulting to the training horizon when unspecified. `clock` selects the progress-clock rule, and `lookback_h` sets its observation window. The bounded conflict correction uses `law=radius`, `q_source=logprob_proxy`, and `refine_optin=true`; `q_star` sets the nominal tolerance and `q_star_momentum_adapt` enables its history adjustment. For refresh decisions, `refresh.tau_u` sets the contrast threshold, `refresh.l_fail` and `refresh.l_imp` set the failure and contrast history windows, and `refresh.beta_fail` set the failure budget rates (Appendix B.4.3).

Runtime outcomes Phase C supplies the selected coefficient for the actor update; in shadow mode, it records the proposal while retaining the configured static coefficient. Invalid conflict inputs or statistics prevent the actor update and leave controller state uncommitted. When refresh control is

disabled, caught controller errors disable the controller for the remainder of the run, and subsequent updates use the static coefficient. With refresh control enabled, caught controller errors stop training. Training also stops if an actor update is skipped after controller-directed bank serving, preventing further updates with inconsistent bank, policy, and controller state.

Recorded decisions The `controller/` metrics record λ_t^{raw} and λ_t, clock positions, cumulative distillation weight, conflict measurements, available history estimates, applied tolerances, and refresh counts by lane. In shadow mode, the cumulative distillation weight sums the proposed weights. Shadow-mode and availability indicators distinguish proposed decisions from applied decisions and missing measurements from zeros. The record is validated before controller state is committed.

C.4 Extension Points

The implementation separates five configurable or extensible interfaces:

1. **Serving policies.** `algorithm.seed.bank.policy` selects the refresh rule. Additional rules extend the serving module's decision function and configuration checks while reusing the bank view and serving loop.
2. **Environment identity.** Environment adapters provide task identities and class taxonomies through the common key interface (Appendix B.1), allowing the bank and controllers to reuse their grouping and lookup logic.
3. **Analyzer models and prompts.** `analysis_backend`, `analysis_model`, and `analysis_prompt_version` select the analyzer and prompt format. Backends return a common skill representation with provenance and accounting information. Prompt-version changes require a compatible bank configuration.
4. **Schedules and tolerances.** The schedule evaluator and refinement functions separate annealing shape from conflict correction. Existing settings support cosine or linear schedules and fixed or history-adjusted tolerances. Additional variants extend these functions and their configuration validation.
5. **Offline analysis.** Skill-version logs, statistics streams, and analysis dumps support refresh, skill-evolution, and recurrence reports. Analysis using these recorded quantities can be added without modifying the training loop.

D Training Setup

D.1 Benchmarks

We use the following three benchmarks to evaluate the model's long horizon agentic capacity:

ALFWorld (Shridhar et al., 2021) renders ALFRED household goals as text-based interactions with $3{,}553$ games, where agents interpret observations, decide appropriate actions, and execute action sequences to reach goals; goals fall into **six** types (*pick*, *pick two*, *look*, *clean*, *heat*, *cool*).

WebShop (Yao et al., 2022) places the agent in a simulated storefront over real product data, turning a natural-language request into search queries, result pages, and attribute inspection before purchase, having $6{,}910$ goals over roughly 50 product category paths.

SearchQA follows the Search-R1 protocol (Jin et al., 2025) and is composed by diverse question sets: an agent gathers evidence through searching over a dense E5 index (Wang et al., 2022a) of the 2018 Wikipedia dump, do reasoning before answering. The training pool contains $169{,}615$ questions, comprising approximately 47% single-hop Natural Questions (Kwiatkowski et al., 2019) and 53% multi-hop HotpotQA (Yang et al., 2018). The full evaluation pool contains $51{,}713$ questions from held-out splits of these two datasets and five additional datasets excluded from our RL training mixture: PopQA (Mallen et al., 2023), 2WikiMultiHopQA (Ho et al., 2020), TriviaQA (Joshi et al., 2017), MuSiQue (Trivedi et al., 2022), and Bamboogle (Press et al., 2023).

D.2 Detailed Setup

Hardware settings. We run the main experiments on **Databricks AI Runtime (AIR)** (Databricks, 2026), using nodes with eight H100 GPUs each. Each ALFWorld or WebShop run uses two training nodes (16 GPUs).

SearchQA additionally uses a dedicated retrieval node, which serves queries over HTTP and does not participate in distributed training. Its 32.3 GB FAISS index is stored in half precision and sharded across the node's eight GPUs. The training allocation depends on policy model size: Qwen3-1.7B and Qwen2.5-3B use one training node (8 GPUs), while Qwen2.5-7B uses four (32 GPUs). Table 11 summarizes the retrieval, training, and total allocations.

Table 11: Hardware allocation per experiment on AIR. Each node contains eight H100 GPUs. Retrieval GPUs serve search queries and are excluded from the training allocation.

		nodes of 8×H100 (H100s)		
Benchmark	**Policy**	**retrieval**	**training**	**total**
ALFWorld, WebShop	all three	—	2 (16)	2 (16)
SearchQA	Qwen3-1.7B, Qwen2.5-3B	1 (8)	1 (8)	2 (16)
SearchQA	Qwen2.5-7B	1 (8)	4 (32)	5 (40)

Ablations run on two local nodes, each with $4\times$H200 GPUs.

Experiment allocation. Due to the budget and time constraints, we run the ablations on local nodes rather than on AIR. For the ablations, we focus on validating the TITRATE design choices and their effects on performance. We also use multiseed runs on the local nodes to assess the stability and TITRATE robustness. Considering that different analyzers may lead to different skill extraction and refinement, we run ablations with multiple analyzers to evaluate the generality of TITRATE across different analyzer models. The main results, on the other hand, are run on AIR with a single analyzer (GPT-4.1) to demonstrate the effectiveness of TITRATE in a production-like setting.

Training horizon. The main experiments use 300 training steps on each benchmark. For ALFWorld and WebShop, each epoch contains one training batch, so 300 epochs correspond to 300 training steps. SearchQA uses an explicit 300-step limit that overrides the epoch-based budget. This SearchQA horizon is supported by a 500-step reference sweep: the three-point-smoothed mean validation score reaches within 1.0 percentage point of its maximum by steps 270, 255, and 255 for the 1.7B, 3B, and 7B policies, respectively. The local component ablations use 320 steps, with a common horizon across the configurations being compared.

Advantage normalization. We retain each benchmark's upstream advantage-normalization setting across all compared methods. ALFWorld and SearchQA center and scale group rewards as in Equation 11; WebShop centers only, using $R_r - \mu_g$. The configuration variable `algorithm.norm_adv_by_std_in_grpo` controls this distinction. These settings affect the scale of the RL gradient and the conflict measurement q_t, so comparisons of these quantities across benchmarks must account for the normalization convention.

RL regularization. In Equation 26, we set $(\kappa_{\mathrm{KL}}, \kappa_{\mathrm{H}}) = (0.01, 0.001)$ on ALFWorld and WebShop and $(0.001, 0)$ on SearchQA; the latter disables entropy regularization.

Decoding. Sampling temperature is 1.0 during training and 0.4 during validation. The Qwen3 model runs with thinking mode *disabled*, while the Qwen2.5 model runs with its default instruct settings. The purpose is to avoid the too long reason traces in thinking model that may exceed the maximum context length and cause too frequent truncation.

Skill-bank configuration. We evaluate task- and class-level keys on ALFWorld and WebShop, and class-level keys on SearchQA. All banked runs use `outcome_split=False` to extract skills jointly from successful and failed trajectories, `skill_evolution=True` to revise existing skills incrementally, and `control_fraction=0.0` to disable held-out control keys.

We select `class_depth` separately for each benchmark based on the resulting groups. ALFWorld uses depth 1, yielding 6 classes and an average of 4.64 usable groups per training step. WebShop and SearchQA use depth 2, yielding 50–52 and 16 classes, respectively, with averages of 3.25 and 4.23 usable groups per step. On ALFWorld, depth 2 produces almost exclusively single-instance groups that yield no common pattern for extraction, motivating the shallower grouping.

Analyzer models. In the main experiments, GPT-4.1 performs skill analysis through an internal gateway supporting up to 16 concurrent requests. In our ablations, we also evaluate Claude Sonnet 4.6, Gemini 2.5 Flash, and GPT-4.1 mini. The analyzer prompt is fixed across all analyzers and all benchmarks.

D.3 Benchmark Settings

Reporting conventions. Additive differences in success rates are expressed in percentage points (pp), and relative changes in %, including the analyzer-call and token reductions in Table 1.

Benchmark settings. Table 12 summarizes the interaction, sampling, and evaluation settings for each benchmark. For advantage normalization, *centre* means subtracting the group mean reward, giving $R_r - \mu_g$; *scale* means additionally dividing by the group standard deviation plus a numerical stabilizer, giving $(R_r - \mu_g)/(\sigma_g + \varepsilon)$ (Equation 11). ALFWorld and SearchQA apply both operations, while WebShop applies centering only.

Table 12: Per-benchmark settings. Every experiment uses these settings; the schedule ablations differ from the main runs in training horizon only, not in any row of this table.

	ALFWorld	**WebShop**	**SearchQA**
episode cap (rollout rounds)	30	15	4
observation history window	5	2	4
train prompts per step	16	16	128
rollouts per prompt	8	8	8
trajectories per step	128	128	1,024
validation episodes	128	128	512
validation interval (steps)	5	5	15
validation points over 300	60	60	20
class-key depth (classes)	1 (6)	2 (∼50)	2 (16)
advantage normalisation	centre, scale	centre only	centre, scale

Episode limits. The configuration variable `env.max_steps` sets the maximum number of interaction rounds per episode: 30 for ALFWorld, 15 for WebShop, and 4 for SearchQA. The trajectory-collection loop stops an episode when the environment signals completion or this limit is reached. These limits remain fixed across the compared methods and their schedule ablations; the adaptive training schedule does not change them.

SearchQA interaction budget. Each episode allows up to four assistant turns, with each search request or final answer consuming one turn. The episode ends when the agent emits an `<answer>` block or reaches the fourth turn. At most three searches are executed: a search request on the fourth turn ends the episode without retrieval. Each search returns up to three passages. At termination, exact-match scoring uses the last `<answer>` block in the accumulated dialogue; if none is present, the reward is zero.

ALFWorld episode budget. We use a 30-round episode limit for ALFWorld, compared with the framework default of 50. All methods share this limit during training and evaluation, so the reported success rates provide a consistent comparison under the same interaction budget. Our comparisons therefore assess performance within this 30-round setting; absolute success rates may differ under longer episode limits.

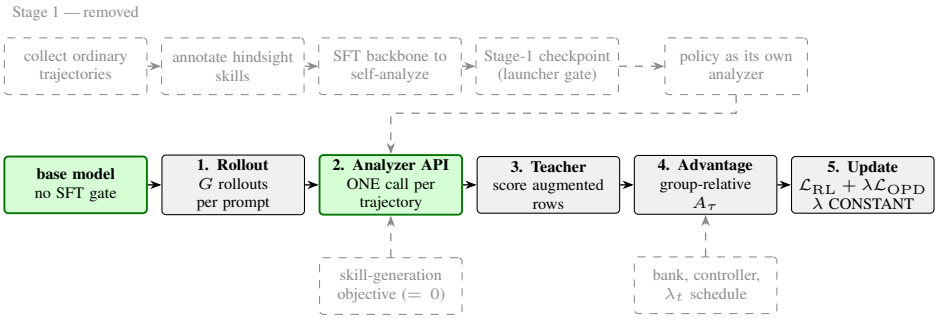


Figure 9: **Adapting SEED for PER-TRAJ.** starts from the original model checkpoint and repeats: (1) collect G rollouts per prompt; (2) extract one skill per trajectory with an external analyzer; (3) compute teacher log-probabilities under skill-augmented context; (4) compute group-relative advantages; and (5) update the policy using $\mathcal{L}_{\mathrm{RL}} + \lambda\mathcal{L}_{\mathrm{OPD}}$ with constant λ. Green boxes show the replacement initialization and analyzer; shaded grey boxes show the retained training operations. Dashed boxes show omitted components, including SEED's preliminary skill SFT and policy-based analyzer, the skill-generation objective, and the bank and adaptive controller used by TITRATE.

D.4 PER-TRAJ Implementation

The **PER-TRAJ** baseline adapts SEED (Wu et al., 2026) to use an external analyzer for per-trajectory skill extraction. It retains skill-conditioned self-distillation with a constant weight, without skill storage or reuse. Figure 9 summarizes the adaptation workflow.

Removing analyzer training. SEED first fine-tunes the policy on hindsight skills so that it can analyze its own trajectories. Because our experiments use an external analyzer, we omit this preliminary SFT stage and initialize training from the original model checkpoint. We also disable the auxiliary skill-generation objective: the policy learns from extracted skills but is not trained to generate them.

Using an external analyzer. For each collected trajectory, the external analyzer extracts an episode-level skill. The teacher then scores the policy's responses under skill-augmented context, providing the self-distillation signal for the actor update. The main experiments use GPT-4.1 as the analyzer and a fixed distillation weight $\lambda_t = 0.01$. PER-TRAJ extracts skills independently for each trajectory, without a skill bank, reuse/refresh decisions, or an adaptive weight schedule. Our observations show that PER-TRAJ's performance fluctuates more than TITRATE's, partly because of 1) the skill extraction focusing on successful or failure experience in isolation, and 2) the constant distillation weight λ_t not adapting to the current training state.

Purpose of the adaptation. PER-TRAJ and the banked methods share the same starting checkpoint, external analyzer, and underlying RL and self-distillation objectives. This avoids introducing differences in preliminary skill training or analyzer identity into the comparison. PER-TRAJ therefore provides a common baseline for evaluating skill management and adaptive distillation weighting.

E Extended Experimental Analysis

E.1 Key Recurrence and Skill-Bank Granularity

A stored skill can be reused or refreshed only when its key appears again. We therefore examine key recurrence over the 300-step training horizon to assess which granularities support these decisions. Table 13 summarizes task-level recurrence under the reported sampling schedule. Repeated encounters indicate opportunities to reuse or refresh previously stored skills.

Task-level recurrence. ALFWorld has $2{,}694$ distinct task keys across $4{,}791$ bank decisions, including $2{,}097$ repeated encounters (43.8%). WebShop has $4{,}212$ keys across $4{,}799$ decisions, including

Table 13: Task-level key recurrence over 300 training steps. Each decision corresponds to a rollout group processed by the bank. Repeat share is the fraction of decisions involving previously encountered keys. The final first-encounter rate is the fraction involving new keys on the last training step.

benchmark	distinct task keys	decisions	repeated encounters	repeat share	final first-encounter rate
ALFWorld	2,694	4,791	2,097	43.8%	0.375
WebShop	4,212	4,799	587	12.2%	0.812
SearchQA	38,400	38,400	**0**	**0.0%**	**1.000**

587 repeated encounters (12.2%). Both benchmarks therefore provide repeated encounters at which a controller can choose between reusing and refreshing a skill.

SearchQA requires broader keys for reuse. Under the reported sampling schedule, all 38,400 SearchQA task keys appear only once. In the task-level diagnostic runs, every bank decision is a first encounter, with no reuse or refresh. Task-level grouping still shares one extracted skill across the rollouts for a question, but offers no reuse across training steps. Consequently, these runs cannot distinguish refresh policies through their reuse or refresh behavior.

At class depth 2, SearchQA instead has 16 recurring keys across 4,301 bank decisions. After the 16 first encounters, the remaining 4,285 decisions provide opportunities for reuse or refresh. We therefore evaluate TITRATE on SearchQA with class-level keys, where its refresh controller can act on previously stored skills.

Implication for key selection. Key granularity should reflect recurrence within the training horizon. Task-level keys support reuse on ALFWorld and WebShop; class-level keys enable it on SearchQA by sharing skills across related questions. Recurrence enables reuse, while the resulting extraction cost and task performance also depend on the grouping and refresh policy. Section E.2 examines these effects.

E.2 Task- versus Class-Level Keys: Extraction Cost and Task Performance

On ALFWorld and WebShop, both task- and class-level keys support reuse (Section E.1). Table 14 compares their success rates and analyzer calls for Qwen3-1.7B over 300 training steps. Results across all three policy models are summarized below.

Table 14: Task- versus class-level keys for Qwen3-1.7B over 300 training steps. Success rate is averaged over the final ten validation points. Δ denotes class minus task in percentage points; Task/Class is the ratio of total analyzer calls under the two granularities.

		Success rate			Analyzer calls		
Benchmark	Method	Task	Class	Δ (pp)	Task	Class	Task/Class
ALFWorld	Bank-Step	0.7062	0.7734	+6.72	4,791	1,654	2.9
	Bank-12	0.6844	0.6500	−3.44	4,568	151	30.3
	Bank-Fail	0.7617	0.8250	+6.33	3,559	236	15.1
	Titrate	0.7867	0.8391	+5.24	3,177	650	4.9
WebShop	Bank-Step	0.5789	0.6188	+3.99	4,799	1,880	2.6
	Bank-12	0.6203	0.6414	+2.11	4,762	431	11.0
	Bank-Fail	0.6256	0.6523	+2.67	4,481	406	11.0
	Titrate	0.6477	0.6606	+1.29	4,308	608	7.1

Cost. Class keying reduces extraction calls in **all 24 experiments** $\times$ **model cells**, by 2.6$\times$ to 57.4$\times$ (the largest being Bank-Fail on Qwen2.5-7B/ALFWorld, 2,925 $\rightarrow$ 51 calls and 78.3 $\rightarrow$ 4.7 M gateway tokens). These are exact counters, not estimates.

Performance. Across the same 24 cells the mean difference is $+0.83$ pp, and the sign is not consistent: class is ahead in 14 cells and behind in 10, and the sign reverses *within* a single model across arms (Qwen2.5-7B on ALFWorld: $+6.25$, -9.61, $+2.18$, -2.27). **Class keying consistently reduces extraction calls, while the direction and size of the success-rate change depend on the configuration.**

Scope of the granularity comparison. The skill key and analyzer prompt change together (`seed_class_group` versus `seed_task_group`), and the configurations also incur different numbers of extractions. The comparison therefore measures class keying as implemented; it does not separate the effects of pooling, prompting, and refresh expenditure.

E.3 Performance Analysis on Benchmarks

E.3.1 Result Analysis from Different Perspectives

Figures 10–12 compare methods through validation success, steps per five-percentage-point improvement, and cumulative analyzer calls on different benchmarks via different policy models. The following paragraphs describe the metrics and their interpretation.

Validation performance. For method a, let $\tau_1 < \cdots < \tau_M \le T$ be the validation steps over the $T = 300$ training-step horizon, and let $z_{a,r}(\tau_i) \in \{0, 1\}$ indicate success on evaluation episode r. With n episodes per evaluation, the success rate is

$$\mathrm{SR}_a(\tau_i) = \frac{1}{n} \sum_{r=1}^{n} z_{a,r}(\tau_i). \tag{59}$$

SearchQA uses exact-match correctness as its binary outcome. Evaluation intervals and sample counts follow Table 12. For $i \ge W$, the plotted trailing mean and standard deviation are defined as follows:

$$\begin{aligned} \overline{\mathrm{SR}}_a(\tau_i) &= \frac{1}{W} \sum_{j=0}^{W-1} \mathrm{SR}_a(\tau_{i-j}), \\ s_a(\tau_i) &= \sqrt{\frac{1}{W} \sum_{j=0}^{W-1} \left[\mathrm{SR}_a(\tau_{i-j}) - \overline{\mathrm{SR}}_a(\tau_i)\right]^2}. \end{aligned} \tag{60}$$

The figures use $W = 10$ on ALFWorld and WebShop and $W = 4$ on SearchQA; both choices span 45 training steps between the earliest and latest readings. Shading shows $\overline{\mathrm{SR}}_a \pm s_a$, describing variation across checkpoints. The main-result summaries use the final ten validation points on every benchmark, including SearchQA.

Steps per five-point success gain. For a success threshold $\gamma \in [0, 1]$, define its first-crossing time as

$$t_a(\gamma) = \min\{\tau_i : i \ge W,\ \overline{\mathrm{SR}}_a(\tau_i) \ge \gamma\}, \qquad \min \emptyset = \infty. \tag{61}$$

Thus, $t_a(\gamma)$ is the first validation step at which the smoothed success rate reaches or exceeds γ; it is infinite if the threshold is never reached within the run. The climbing figure reports these times directly. For the step-cost matrices, finite crossing times are rounded up to the next multiple of g, with $g = 10$ on ALFWorld and WebShop and $g = 15$ on SearchQA. For band starts $\gamma \in [0, 0.95]$, we report a cost only when the upper threshold is reached within the run and exceeds the first smoothed success rate $\overline{\mathrm{SR}}_a(\tau_W)$. For these bands,

$$\begin{aligned} \widetilde{t}_a(\gamma) &= g \left\lceil \frac{t_a(\gamma)}{g} \right\rceil, \\ C_a(\gamma) &= \widetilde{t}_a(\gamma + 0.05) - \widetilde{t}_a(\gamma). \end{aligned} \tag{62}$$

Other cells are blank; a reported zero means both thresholds fall in the same time bin. If the first smoothed observation lies inside a band, its cost is retained and measures time from that observation, rather than a full five-point gain.

Example and interpretation. For TITRATE on ALFWorld/Qwen3-1.7B, $t_a(0.75) = 250$ means that step 250 is the first evaluation with smoothed success of at least 75%. Since $t_a(0.80) = 290$, both times are already multiples of 10, and $C_a(0.75) = 290 - 250 = 40$ training steps. This records first attainment: the curve falls below 80% by step 300, and a twenty-point window does not reach 80%. The matrix should therefore be read alongside the success curve; fast progress through lower thresholds need not imply higher final performance.

Analyzer cost. With $c_a(u)$ counting initial-extraction and refresh requests at step u regardless of outcome, the panels plot the cumulative count on a logarithmic axis:

$$\mathrm{Calls}_a(t) = \sum_{u=1}^{t} c_a(u). \tag{63}$$

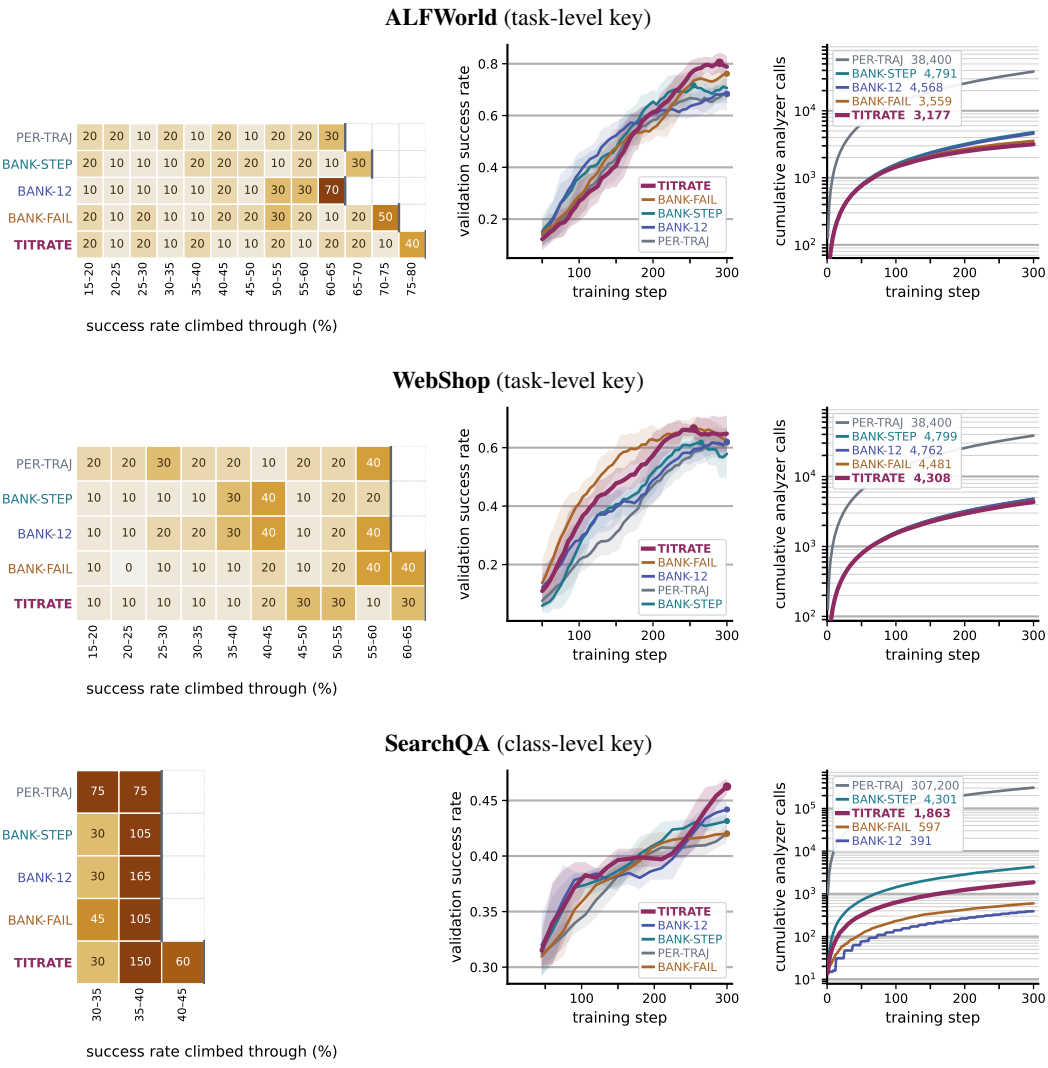
E.3.2 Different Models on Different Benchmarks

Figures 10–12 give one figure per model, with three benchmarks as rows, showing how model performance across benchmarks on different methods.

The benchmark sets the operating range; the model sets the headroom. The three benchmarks occupy different success-rate ranges — 68–92% on ALFWorld, 58–78% on WebShop and 41–51% on SearchQA across all arms and models — so they are three regimes rather than three points on a difficulty axis, and we never pool them. Within a benchmark, scale mostly buys level rather than ordering: ALFWorld rises from 68–79% at Qwen3-1.7B to 84–92% at Qwen2.5-7B, and SearchQA from 41–43% to 48–51%. What changes with it is the room left for the refresh policy to matter. The five arms span 10.5 pp on ALFWorld at Qwen3-1.7B but only 2.2 pp at Qwen2.5-3B, and on SearchQA they span 1.8–3.4 pp at every scale. Figure 13 shows the same compression in the ceilings each arm reaches: four distinct ceilings on ALFWorld at Qwen3-1.7B, two at the larger models, and on WebShop at Qwen2.5-7B all five arms finish within the same five-percentage-point success-rate interval. **Method choice separates the arms where the model is weakest relative to the task, and the separation closes as either the model or the benchmark becomes easy.**

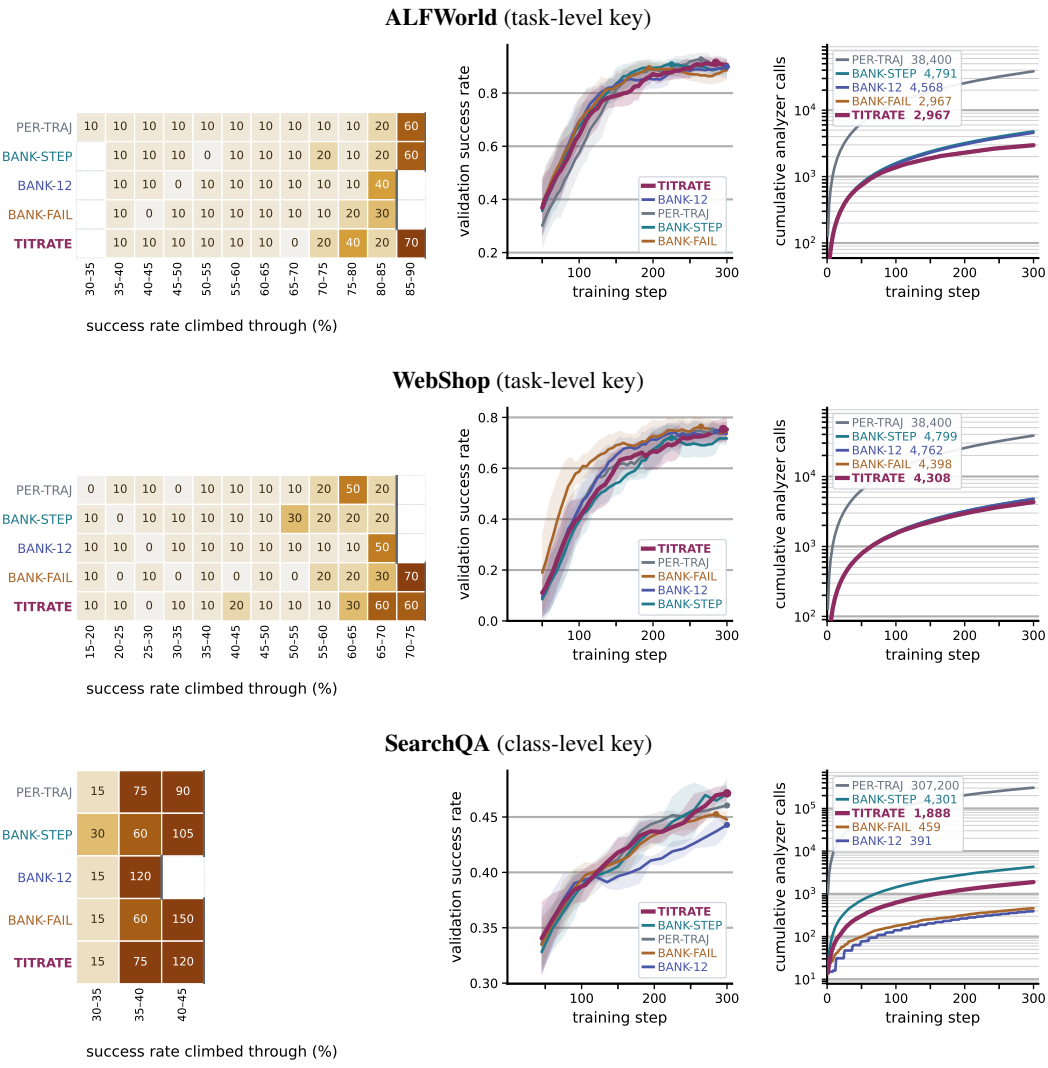
TITRATE's success margins vary across models and benchmarks. Against the strongest baseline in each setting, TITRATE's mean-success differences are $+2.50$, $+2.21$, and $+0.93$ pp for Qwen3-1.7B (ALFWorld, WebShop, SearchQA); $+0.92$, $+0.61$, and -0.24 pp for Qwen2.5-3B; and $+1.57$, $+0.51$, and $+2.42$ pp for Qwen2.5-7B. TITRATE has the highest observed mean success in eight of nine settings; the exception is SearchQA/Qwen2.5-3B, where BANK-STEP leads by 0.24 pp.

Analyzer-call reductions across benchmarks. TITRATE uses $1/12$–$1/13$ as many analyzer calls as PER-TRAJ on ALFWorld, $1/8.9$ on WebShop, and $1/163$–$1/165$ on SearchQA. On SearchQA, BANK-12 costs less (391 calls) but achieves 1.24–3.41 pp lower success. Its fixed waiting period and BANK-FAIL's all-failure requirement can suppress updates despite unresolved failures (Section E.3.4). TITRATE selectively admits mixed-outcome updates and achieves higher observed success than both (Section E.3.5).



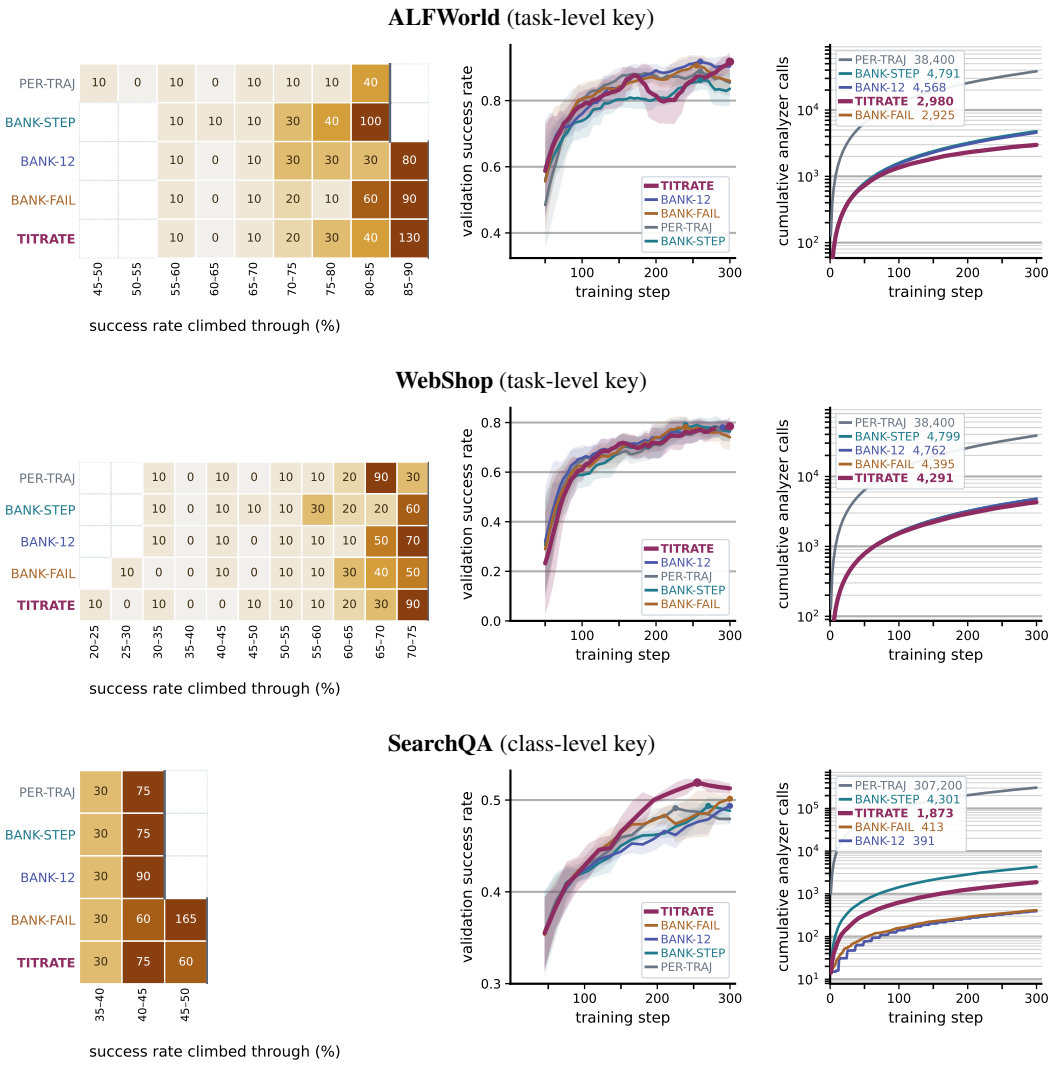
(a) Steps per 5-point success gain. (b) Val Success Rate. (c) Analyzer Calls Volume.

Figure 10: Qwen3-1.7B: TITRATE vs baselines. Columns, left to right: steps per 5-point success gain, validation success rate, cumulative analyzer calls. On ALFWorld TITRATE ends highest (78.9%) and cheapest (3, 177 calls); on WebShop TITRATE ends highest (64.8%) and cheapest (4, 308 calls); on SearchQA it ends highest (46.3%) for 1, 863 calls, against BANK-12's 391.



(a) Steps per 5-point success gain. (b) Val Success Rate. (c) Analyzer Calls Volume.

Figure 11: Qwen2.5-3B: TITRATE vs baselines. Columns, left to right: steps per 5-point success gain, validation success rate, cumulative analyzer calls. On ALFWorld it ends highest (90.7%) for 2,967 calls, against BANK-FAIL's 2,967; on WebShop TITRATE ends highest (75.3%) and cheapest (4,308 calls); on SearchQA it ends highest (47.1%) for 1,888 calls, against BANK-12's 391.



(a) Steps per 5-point success gain. (b) Val Success Rate. (c) Analyzer Calls Volume.

Figure 12: Qwen2.5-7B: TITRATE vs baselines. Columns, left to right: steps per 5-point success gain, validation success rate, cumulative analyzer calls. On ALFWorld it ends highest (91.7%) for 2,980 calls, against BANK-FAIL's 2,925; on WebShop TITRATE ends highest (78.5%) and cheapest (4,291 calls); on SearchQA it ends highest (51.3%) for 1,873 calls, against BANK-12's 391.

E.3.3 CLIMBING PERFORMANCE ON QWEN3-1.7B

Figure 13 compares the same methods on the three benchmarks using the first-crossing times $t_a(\gamma)$ from Equation 61. For the displayed five-percentage-point thresholds $\mathcal{G}$, the highest attained level is

$$\gamma_a^{\max} = \max\{\gamma \in \mathcal{G} : t_a(\gamma) < \infty\}. \tag{64}$$

If no threshold in $\mathcal{G}$ is reached, $\gamma_a^{\max}$ is undefined.

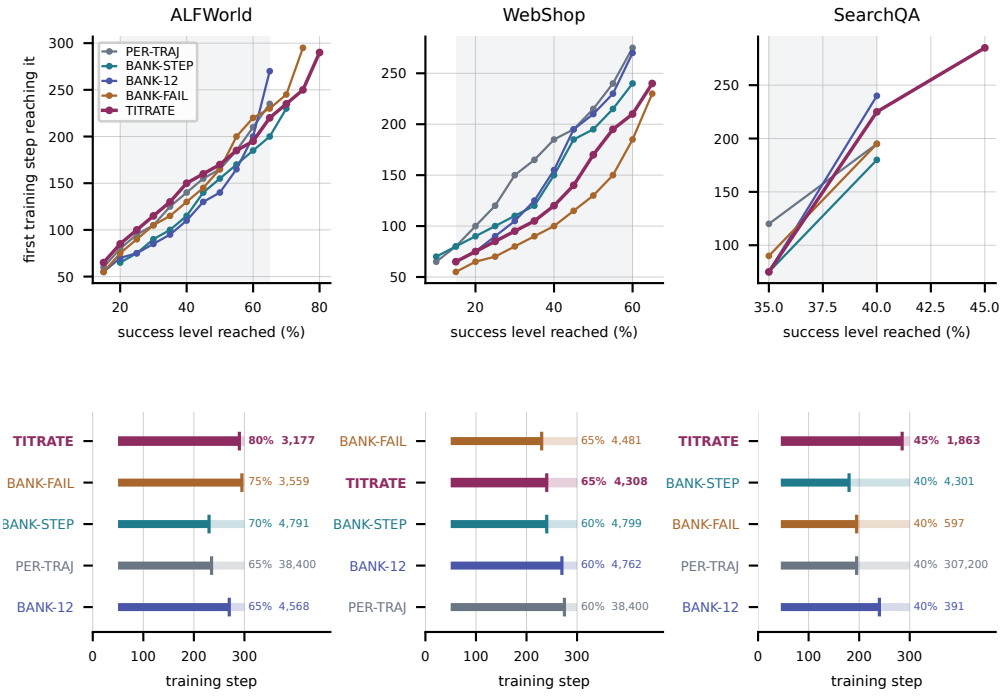


Figure 13: Qwen3-1.7B climbing behaviour: when each arm reaches a success level (top) and how far it gets, and when, before plateauing (bottom).

How to read it. The **top** row plots the first training step at which an arm's trailing validation mean reaches a given success level, so lower is earlier and the slope is what the next five points cost. Each curve stops at that arm's own ceiling: a short curve is an arm that stopped climbing, not a truncated plot, and the shaded span marks the levels every arm crossed. The **bottom** row gives one lane per arm — the solid bar runs from its first measurable point to the step it reached its highest success-rate threshold on the five-percentage-point grid, the faded tail shows the remaining training steps without reaching a higher displayed threshold, and the label gives that ceiling and the arm's analyzer-call total. Lanes are ranked by ceiling, then by time to reach it, then by analyzer cost; on SearchQA PER-TRAJ and BANK-FAIL both reach 40% at step 195, and that last key separates $307{,}200$ calls from 597. Plotting the destination rather than a rate over the levels every arm crossed is deliberate: a common-range rate forgives an arm for stopping early, and on ALFWorld PER-TRAJ and BANK-12 stall at 60–65% while TITRATE reaches 75–80%.

What it shows. On ALFWorld the ceilings separate four ways and TITRATE is alone in the 75–80% success-rate interval, three five-percentage-point intervals above PER-TRAJ, which plateaus for its last 65 steps. On WebShop PER-TRAJ is the latest arm at eight of the ten levels and still ends one five-percentage-point success-rate interval below TITRATE and BANK-FAIL. On SearchQA only TITRATE clears 40%. The pattern common to all three is that the PER-TRAJ arm reaches a ceiling and stops there, having spent 9–$165\times$ TITRATE's analyzer calls to do so.

Temporal resolution. The trailing mean uses 10 validation readings spaced 5 training steps apart on ALFWorld and WebShop, and 4 readings spaced 15 steps apart on SearchQA. Both windows span 45 steps and summarize observations centered 22.5 steps before their timestamp. Crossing times follow the validation cadence: the recorded 240- and 245-step ALFWorld climbs differ by one evaluation interval, while SearchQA crossing times advance in 15-step increments.

E.3.4 Refresh Behavior under Class-Level Pooling

SearchQA class keys recur frequently: the median revisit gap is one training step. Low refresh counts therefore reflect the baseline trigger rules rather than a lack of opportunities to revisit stored skills.

For the following diagnostics, a marked serving has mixed outcomes, a success fraction no higher than at its preceding same-key serving, and remaining failure mass $E_k(t) > 0.05$. These retrospective marks identify non-improving mixed outcomes; they do not establish that a refresh would improve performance.

A fixed interval delays updates. BANK-12 reuses a skill on 3,910 of 4,285 existing-key servings (91.25%) because its age is below twelve steps. This includes 91.57–91.87% of marked servings. Almost all marks with complete follow-up nevertheless receive a refresh at that serving or within the next twelve steps (Table 15). The restriction is therefore a waiting period rather than persistent exclusion.

Class-wide failure can hide failed tasks. BANK-FAIL requires all rollouts in the previous class serving to fail. Success on one task can therefore prevent refresh despite complete failure on another. This occurs on 74.05–74.52% of revisits. Among eligible marked servings, 68.73–72.08% receive no refresh within twelve subsequent steps, and three to five class keys are never refreshed after creation despite appearing throughout training. The pattern extends across keys and remains visible with longer histories of non-improving success.

Table 15: Baseline trigger diagnostics on SearchQA over 300 training steps. Panel (a) separates retrospective marked servings from cross-task suppression based on the previous completed serving. All marked BANK-FAIL servings reuse by construction; their prevalence is reported rather than a 100% exclusion rate. Panel (b) includes marked servings at steps 1–288, each with 12 subsequent steps observed. A refresh at the marked serving counts as delay zero; the denominator includes all eligible marks, not only those initially reused. The final column counts keys with no refresh attempt after initial extraction during the observed run.

(a) Exposure to the baseline trigger restrictions

Model	BANK-12: reused / marked servings	BANK-FAIL: cross-task suppression / revisits	BANK-FAIL: marked / revisits
Qwen3-1.7B	1,520/1,655 (91.84%)	3,189/4,285 (74.42%)	1,655/4,285 (38.62%)
Qwen2.5-3B	1,594/1,735 (91.87%)	3,193/4,285 (74.52%)	1,736/4,285 (40.51%)
Qwen2.5-7B	1,630/1,780 (91.57%)	3,173/4,285 (74.05%)	1,749/4,285 (40.82%)

(b) Subsequent refresh timing and keys with no refresh after creation

Model	BANK-12: no refresh within 12 steps	BANK-FAIL: no refresh within 12 steps	BANK-FAIL: keys with no refresh in 300 steps
Qwen3-1.7B	1/1,584 (0.06%)	1,086/1,580 (68.73%)	3/16
Qwen2.5-3B	5/1,662 (0.30%)	1,172/1,666 (70.35%)	5/16
Qwen2.5-7B	4/1,701 (0.24%)	1,208/1,676 (72.08%)	4/16

The effect depends on key granularity. Task keys recur less frequently on ALFWorld and WebShop: the Qwen2.5-7B histories have median revisit gaps of 67 and 83 steps. Consequently, most revisits already satisfy BANK-12's age threshold. Separate task keys also prevent success on one task from suppressing another task's refresh, although mixed outcomes within a task still disable BANK-FAIL. Table 16 summarizes these differences.

Table 16: Refresh attempts per existing-key serving over 300 steps. Initial counts first extractions; Existing counts $R + U$. These counts and BANK-12's refresh rate are identical across the three models. BANK-FAIL ranges show the minimum and maximum across models. SearchQA task keys never recur, so their refresh rate is undefined. These are bank-decision statistics, not total gateway-request counts.

Benchmark	Key	Initial	Existing	BANK-12 (%)	BANK-FAIL (%)
ALFWorld	task	2,694	2,097	89.37	11.02–41.25
	class	6	1,648	8.80	2.73–13.96
WebShop	task	4,212	587	93.70	31.18–45.83
	class	42	1,838	21.16	12.02–19.80
SearchQA	task	38,400	0	—	—
	class	16	4,285	8.75	9.26–13.56

E.3.5 TITRATE Admits Mixed-Outcome Updates Selectively

TITRATE's contrast lane can refresh a partially successful class when progress stalls and sufficient failure mass remains. On SearchQA, TITRATE refreshes 43.10–43.69% of existing-key servings, compared with 8.75% for BANK-12 and 9.26–13.56% for BANK-FAIL. Contrast accounts for 88.96–94.88% of TITRATE's refresh attempts, while only 46.76–48.67% of contrast candidates pass admission (Table 17). All admitted candidates are selected in these runs, so the observed filtering comes from the admission conditions.

Table 17: Executed TITRATE refresh decisions on SearchQA, summed over steps 1–300. Candidates are decidable existing-key servings in each lane before admission; first encounters and solved servings are excluded. Selected counts analyzer refresh attempts, not committed updates; every admitted candidate was selected. Contrast share divides contrast selections by all refresh attempts. The Qwen2.5-7B run has one unsuccessful attempt whose lane cannot be identified from these aggregate counters.

	Failure		Contrast		Contrast
Model	Candidates	Selected	Candidates	Selected	share (%)
Qwen3-1.7B	599	204	3,514	1,643	88.96
Qwen2.5-3B	442	106	3,667	1,766	94.34
Qwen2.5-7B	395	95	3,620	1,762	94.88

These observations explain how the refresh rules behave under class pooling. They demonstrate TITRATE's use of an additional update route, but do not isolate its contribution to final accuracy or imply that every skipped refresh was necessary.

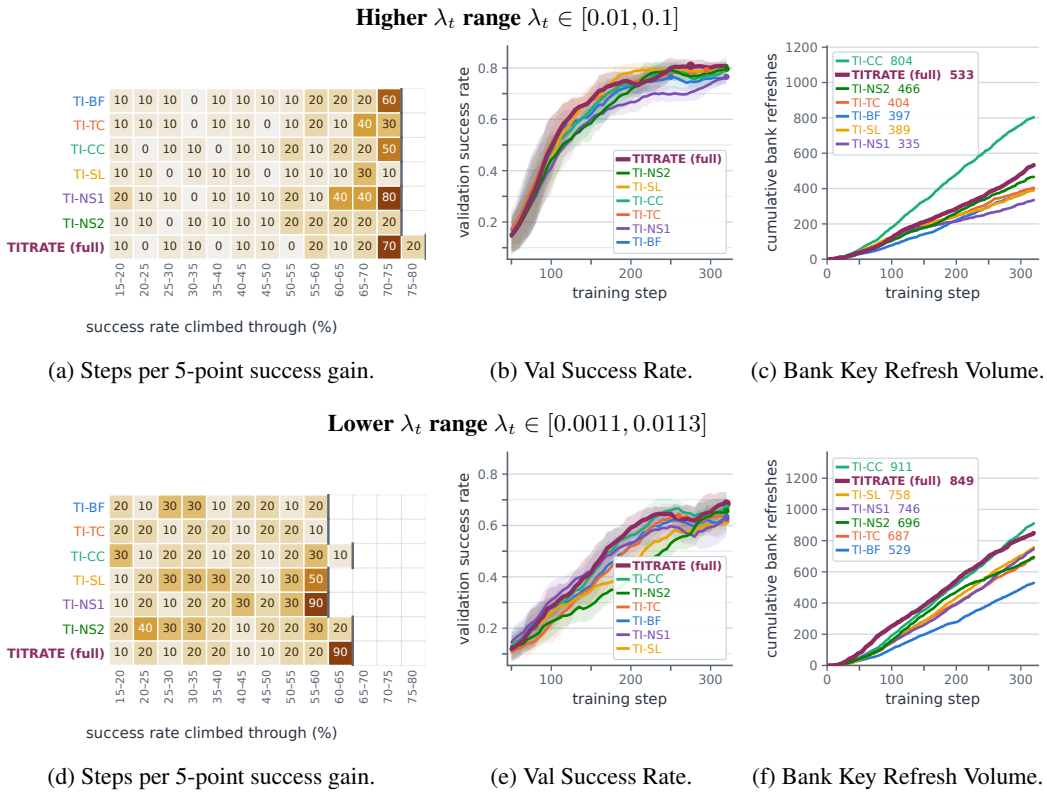
F More Ablations

F.1 Component Effects on Learning Dynamics and Refresh Cost

Figure 14 compares learning progress and cumulative skill refreshes on ALFWorld / Qwen3-1.7B with task-level keys. Full TITRATE achieves the highest final mean success in both λ_t ranges, although it is neither uniformly fastest at reaching intermediate thresholds nor the method with the fewest refreshes.

Early progress does not determine final performance. In the higher λ_t range, the linear schedule (TI-SL) initially advances faster than TITRATE but finishes below it. The threshold matrix likewise shows that TITRATE does not minimize the time spent on every five-point gain. These local crossing times describe progress between thresholds; the validation trajectories show whether that progress is sustained.

Schedule choices matter more in the lower range. The lower λ_t range produces lower final success across all configurations. TI-SL and TI-NS2 also lag TITRATE through much of the early



(a) Steps per 5-point success gain. (b) Val Success Rate. (c) Bank Key Refresh Volume.

(d) Steps per 5-point success gain. (e) Val Success Rate. (f) Bank Key Refresh Volume.

Figure 14: Component ablation on ALFWorld / Qwen3-1.7B, task-level key. Top row: the higher λ_t range $\lambda_t \in [0.01, 0.1]$; bottom row: the lower λ_t range $\lambda_t \in [0.0011, 0.0113]$. The experiments using these two λ_t ranges are analyzed separately. Columns, left to right: steps per 5-point success gain, validation success rate, cumulative bank refreshes.

and middle training period. Their final success deficits increase from 1.13 to 7.04 pp and from 1.04 to 2.85 pp, respectively (Table 2), suggesting that schedule shape and Stage 2 refinement become more consequential when distillation weights are smaller.

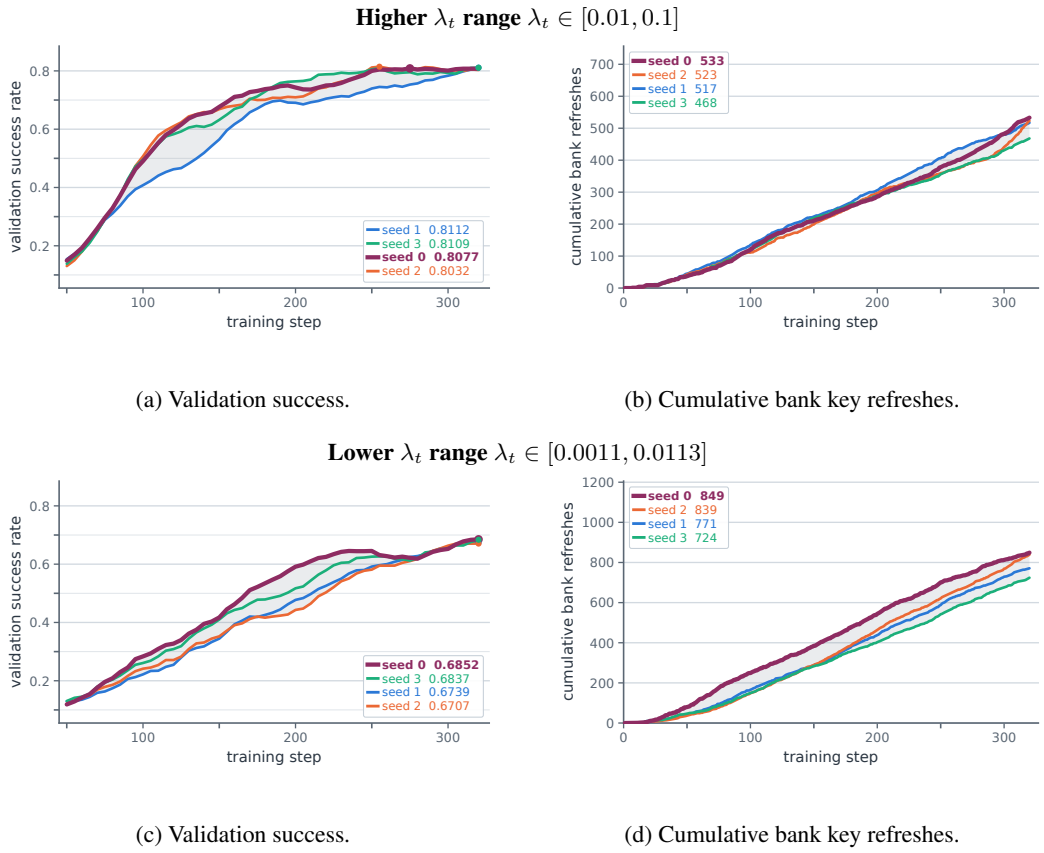
Refresh volume alone does not explain performance. The compensatory policy (TI-CC) accumulates the most refreshes in both ranges without matching TITRATE's final success. Conversely, reducing failure or contrast sampling saves refreshes but also lowers success. Together, these observations support selective refresh admission: increasing refresh volume alone is insufficient, while suppressing refreshes can remove useful learning opportunities.

G Further Discussions

G.1 Multiple Runs and Variance Under Different Seeds

TITRATE yields consistent final performance across environment seeds in both λ_t ranges, despite variation in skill refresh counts. Across four seeds, final success spans only 0.80 pp in the higher range and 1.45 pp in the lower range, with across-seed means of 0.8082 ± 0.0037 and 0.6784 ± 0.0072, respectively (mean $\pm$ sd; Table 18). Thus, refresh-count variation accompanies only minor differences in final performance.

We repeat full TITRATE on ALFWorld / Qwen3-1.7B with four environment seeds (`env.seed`=0–3), task-level keys, and 320 training steps per run, keeping the remaining settings fixed within each λ_t range. Seed 0 is the full TITRATE run in Table 2. Figure 15 shows validation success and cumulative refreshes.



(a) Validation success.

(b) Cumulative bank key refreshes.

(c) Validation success.

(d) Cumulative bank key refreshes.

Figure 15: TITRATE across four environment seeds on ALFWorld / Qwen3-1.7B with task-level keys. Rows show the higher and lower λ_t ranges. Left: trailing-10 validation success (dots: peaks; legends: final means). Right: cumulative committed refreshes through step 320. At each training step, the grey band spans the lowest to highest values across the four seeds. The thicker purple curve shows seed 0, whose results are reported as full TITRATE in Table 2.

Table 18: TITRATE across four seeds in both λ_t ranges on ALFWorld / Qwen3-1.7B with task-level keys. Per-seed success is the final-10 validation mean $\pm$ sd; summary rows report across-seed mean $\pm$ sd and coefficient of variation (cv). Refresh and Analysis Call count committed refreshes and analyzer calls through step 320. Seed 0 reproduces Table 2.

	Higher range $\lambda_t \in [0.01, 0.1]$			**Lower range** $\lambda_t \in [0.0011, 0.0113]$		
seed	**Success Rate** $\uparrow$	**Refresh**	**Analysis Call** $\downarrow$	**Success Rate** $\uparrow$	**Refresh**	**Analysis Call** $\downarrow$
0 (Table 2)	0.8077±0.0170	533	3,320	0.6852±0.0466	849	3,636
1	0.8112±0.0193	517	3,316	0.6739±0.0171	771	3,549
2	0.8032±0.0100	523	3,272	0.6707±0.0195	839	3,629
3	0.8109±0.0395	468	3,252	0.6837±0.0368	724	3,528
across-seed mean±sd	0.8082±0.0037	510±29	3,290±33	0.6784±0.0072	796±59	3,586±55
across-seed cv	0.46%	5.67%	1.01%	1.06%	7.42%	1.53%

Seed variation is smaller than the observed ablation effects. The success losses from component ablations are 1.05–4.75 pp in the higher range and 1.57–7.04 pp in the lower range (Table 2), exceeding the corresponding across-seed spreads of 0.80 and 1.45 pp. These comparisons support the ablation results, but confirming the smaller performance gaps requires repeating the ablated variants across seeds.

Refresh counts vary more than final success. The higher- and lower-range runs commit 510 ± 29 and 796 ± 59 refreshes, respectively, with coefficients of variation of 5.7% and 7.4%, compared with

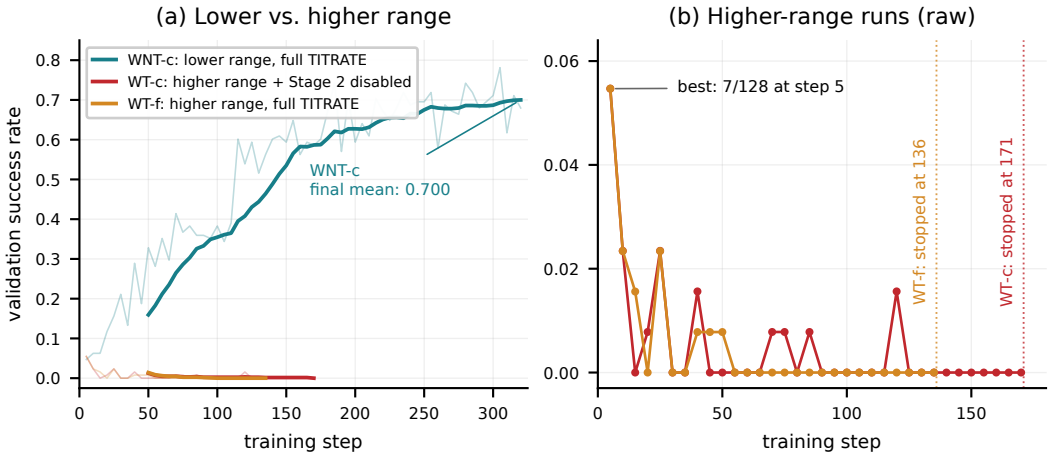


Figure 16: WebShop / Qwen3-1.7B validation success with task-level keys. WNT-c: lower range $[0.0021, 0.0213]$, full TITRATE with Stage 2; WT-c: higher range $[0.01, 0.1]$ with Stage 2 disabled; WT-f: higher range, full TITRATE with Stage 2. Left: faint curves show individual evaluations; thick curves show the average of the latest 10 evaluations. Right: individual higher-range evaluations, magnified. WNT-c completes 320 steps; WT-c and WT-f stop at 171 and 136, respectively. Curves end at the last recorded evaluation.

0.46% and 1.06% for final success. Total analyzer calls vary less (1.0% and 1.5%), as roughly 2,787 first-encounter calls dominate each run's total. Although validation trajectories diverge during training, they converge to similar final success within each range (Figure 15). Final performance remains similar across seeds even when refresh counts differ. However, whether the small performance differences are solely caused by refresh counts remains to be determined.

G.2 Benchmark Sensitivity to the Distillation Weight Range

To study how the distillation weight range affects model performance, we compare three configurations on WebShop / Qwen3-1.7B: **WNT-c** and **WT-f** use full TITRATE at the lower λ_t range $[0.0021, 0.0213]$ and higher range $[0.01, 0.1]$ respectively, while **WT-c** uses the higher range with Stage 2 (Section 3.3) disabled.

The distillation-weight range that works on ALFWorld can cause performance collapse on WebShop. On WebShop / Qwen3-1.7B, the lower range $\lambda_t \in [0.0021, 0.0213]$ completes 320 steps and reaches 0.7000 ± 0.0447 validation success over the final 10 evaluations (mean $\pm$ sd). The higher range $[0.01, 0.1]$ instead collapses to zero success (Figure 16).

Performance under the lower range. WNT-c successfully improves from $6/128$ success at step 5 to a peak of $100/128$ at step 305, with a final-10 mean of 0.7000.

Early validation collapse under the higher range. WT-c and WT-f both reach their highest validation success at the first evaluation ($7/128$ at step 5), then record zero success throughout their final 10 evaluations before stopping. In WT-c, mean response length falls to 11 tokens by step 67, and KL loss peaks at 3.01. By contrast, WNT-c maintains mean response lengths of 181–254 tokens, with a maximum KL loss of 0.41. WT-c's realized distillation coefficient remains above 0.0528, exceeding the lower range's upper bound of 0.0213. Together with the strong ALFWorld results obtained using the same higher range (Table 2), these observations suggest that suitable λ_t ranges depend on the benchmark. We leave systematic range selection and benchmark-adaptive tuning to future work.

G.3 Comparison of Analyzer Models

We compare GPT-4.1 (reference), Claude Sonnet 4.6, Gemini 2.5 Flash, and GPT-4.1 mini as TITRATE's analyzer on ALFWorld/Qwen3-1.7B, using the same configuration as we did in Sec-

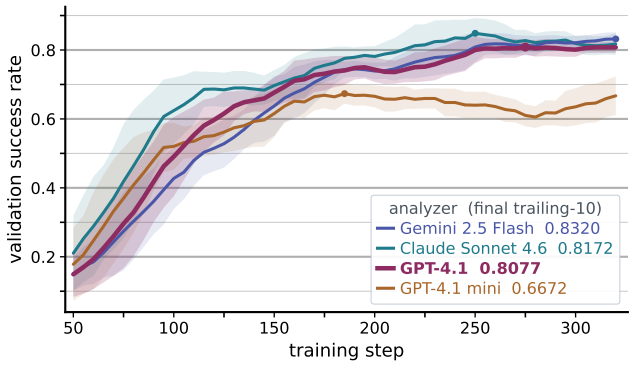


Figure 17: Validation success for four analyzers on ALFWorld/Qwen3-1.7B. Curves show the final-10 mean, ribbons ± 1 sd within that window, and dots each curve's peak. Legends report final trailing means. GPT-4.1 mini plateaus after step ≈ 160.

Table 19: Analyzer comparison. Success Rate is the final-10 validation mean $\pm$ sd; Refresh and Analysis Call are total committed refreshes and analyzer calls (changes relative to GPT-4.1). Cum. weight is $\sum_t \lambda_t$; p50 latency is the median over steps of per-step median analyzer response times. Bold indicates success above the reference, without establishing a ranking.

Analyzer	Success Rate ↑	Refresh	Analysis Call ↓	Cum. weight	p50 latency
GPT-4.1 (reference)	0.8077±0.0170	533	3,320	15.02	2.9 s
Claude Sonnet 4.6	0.8172±0.0332	378 -29.1%	3,165 -4.7%	14.94	6.6 s
Gemini 2.5 Flash	0.8320±0.0203	433 -18.8%	3,220 -3.0%	15.00	1.9 s
GPT-4.1 mini	0.6672±0.0591	601 +12.8%	3,388 +2.0%	15.03	2.6 s

tion 4.3: $\lambda_t \in [0.01, 0.1]$, task-level keys, and 320 training steps. Prompts, concurrency, and controller settings are kept the same; the two non-OpenAI models set larger prompt budgets to retain complete trajectories, and Gemini's reasoning model is disabled. All calls return parseable structural response during experiments.

Analyzer capability limits late-stage gains. The three frontier analyzer models finish at 0.8077–0.8320 success rate, whereas GPT-4.1 mini reaches 0.6672, 14.05 pp below the reference (Table 19). Claude and Gemini exceed GPT-4.1 by 0.95 and 2.43 pp, respectively. GPT-4.1 mini shows substantially lower late-stage performance in this comparison. Despite leading the reference over the first 80 steps (0.298 versus 0.248 mean success), it plateaus near 0.65 after step ≈ 160, while the others reach 0.81–0.83 (Figure 17). This pattern suggests that analyzer capability matters most for resolving residual failures later in training, which motivates us to adapt GPT-4.1 as the analyzer for TITRATE.

Refresh demand and latency distinguish analyzer costs. Relative to GPT-4.1, GPT-4.1 mini records 12.8% more refreshes despite lower success, while Claude Sonnet 4.6 and Gemini 2.5 Flash record 29.1% and 18.8% fewer, respectively. All four runs share 2,787 first-encounter calls, so the corresponding changes in total analyzer calls are smaller: $+2.0\%$, -4.7%, and -3.0%. Claude Sonnet 4.6 uses fewer calls but has the highest median response latency (6.6 s), compared with 2.9 s for GPT-4.1, 2.6 s for GPT-4.1 mini, and 1.9 s for Gemini 2.5 Flash. GPT-4.1 provides a reasonable balance of observed task performance and response latency: it substantially outperforms GPT-4.1 mini while responding faster than Claude Sonnet 4.6. This supports its use as our reference analyzer, although Gemini 2.5 Flash also offers a promising alternative, with higher observed success and lower response latency in this comparison.