OFFLINE GUIDANCE, ONLINE REASONING: REUSING LLM FEEDBACK FOR SMALL LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) offer strong reasoning capabilities but are often costly to access through commercial APIs, while small language models (SLMs) are easier to deploy locally yet remain weaker in reasoning. This capability–deployment gap has motivated LLM–SLM collaboration, which aims to improve SLM reasoning using LLM capabilities while preserving the deployment advantages of SLMs. Existing approaches mainly follow two paradigms. Knowledge distillation uses LLM-generated answers and reasoning trajectories to train SLMs offline, but requires parameter updates and additional training. Alternatively, online collaboration routes difficult problems to an LLM or leverages LLM-generated guidance and corrections when an SLM encounters difficulties. Although effective, online collaboration requires repeated LLM access. Moreover, the guidance produced for a particular problem is discarded after inference and cannot benefit subsequent problems involving similar reasoning states. In the paper, we focus on a more constrained setting in which the LLM is accessed only offline, the SLM parameters remain fixed, and online inference is performed solely by the SLM. To this end, we propose **Reusable Latent Correction (RLC)**, which converts one-off natural-language guidance from a black-box LLM into persistent corrective experiences in the hidden space of an SLM. RLC stores these experiences in an external bank and retrieves them according to the SLM's current reasoning state, enabling the SLM to reuse LLM-derived corrections during inference without any online LLM calls. Experiments across multiple reasoning benchmarks and SLM scales show that RLC consistently improves SLM reasoning without parameter updates or online LLM calls. Code is available at `https://anonymous.4open.science/r/anonymous-review-code-FBE5/`.

1 INTRODUCTION

In recent years, large language models (LLMs) have made remarkable progress on tasks such as mathematical reasoning (Hendrycks et al., 2021; Yang et al., 2024) and complex question answering (Rein et al., 2024). However, strong reasoning capabilities typically rely on frontier LLMs at massive scales (OpenAI, 2023; Guo et al., 2025; Yang et al., 2025). Most of these LLMs are accessible only through commercial APIs, leaving their parameters and internal states inaccessible. By contrast, small language models (SLMs) with 3B–14B parameters offer flexible deployment, low inference costs, and access to their internal states. Nevertheless, due to their limited model capacity, their reasoning capabilities still fall behind those of frontier LLMs. It is therefore of practical importance to investigate how LLMs can be leveraged to improve the reasoning capabilities of SLMs while preserving the latter's deployment advantages.

To this end, various paradigms for collaboration between LLMs and SLMs have been proposed. As shown in Figure 1, these approaches mainly follow two directions. The first employs knowledge distillation (Figure 1(a)), using answers and chain-of-thought (CoT) trajectories generated by an LLM to train an SLM **offline**, thereby transferring part of the teacher's knowledge to the student (Magister et al., 2023; Hsieh et al., 2023; Zhu et al., 2024). However, such methods typically require updating the parameters of the SLM, which introduces additional training costs. As shown in Figure 1(b), the second direction introduces model routing or LLM–SLM collaboration during **online inference**: the system delegates difficult problems to an LLM or requests guidance and corrections from it when the SLM fails (Chen et al., 2024; Ong et al., 2025; Lee et al., 2026; Zeng et al., 2026; Kim et al., 2025).

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.

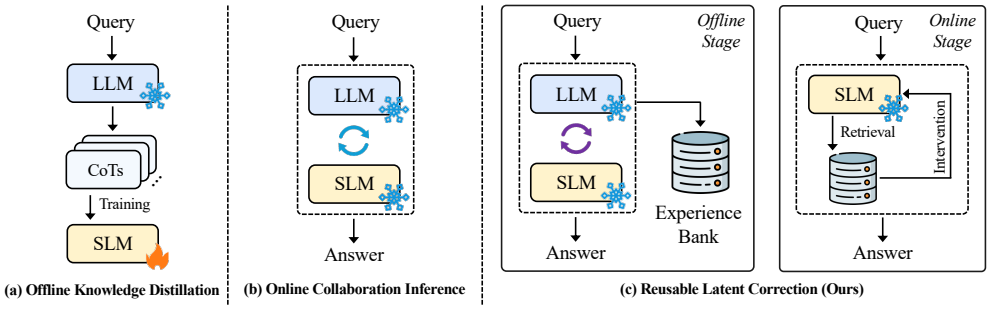


Figure 1: Comparison of LLM–SLM collaboration paradigms. **(a)** Offline knowledge distillation trains the SLM with LLM-generated CoT trajectories, requiring parameter updates. **(b)** Online collaboration avoids retraining but repeatedly queries the LLM during inference. **(c)** RLC uses the LLM only for offline experience construction and stores effective corrections in an external Bank. At inference time, the frozen SLM retrieves relevant corrections and applies hidden-state interventions without further LLM access.

Although these methods can directly exploit the reasoning capabilities of LLMs, they depend on continued online access to them. Besides, the guidance provided by the LLM is typically used only for the current inference instance. Once the task is completed, this valuable guidance disappears with the context and cannot benefit subsequent problems that encounter similar reasoning states.

These two paradigms exploit LLM capabilities by either updating the SLM's parameters or repeatedly invoking the LLM. In practical applications, however, we would like to avoid both costs. Specifically, we consider a constrained collaboration setting involving a black-box LLM accessible only through an API and capable of providing natural-language feedback, together with a locally deployable SLM whose internal states are accessible. Our goal is to invoke the LLM only during an offline stage and, without updating the parameters of the SLM, transform its guidance into persistent reasoning experiences. During online inference, the SLM should then solve problems independently, without further access to the LLM. This raises a central research question:

Can natural-language guidance provided by a black-box LLM for a specific reasoning error be transformed into an internal corrective experience that an SLM can reuse on future problems?

To address this question, as shown in Figure 1(c), we propose **Reusable Latent Correction (RLC)**, a framework for improving SLM reasoning by reusing experiences acquired from an LLM. RLC transforms the LLM from an online solver into an offline experience provider and converts its one-off natural-language guidance into persistent corrective experiences in the hidden space of the SLM. These experiences are stored explicitly in an external experience bank. During online inference, RLC retrieves and injects relevant corrective experiences on demand according to the current state of the SLM, without invoking the LLM. Specifically, the framework consists of two stages: offline experience construction and online experience reuse.

During offline experience construction, the SLM first generates a step-by-step reasoning process for a given problem. Model uncertainty is then used to identify candidate steps that may cause subsequent reasoning to fail. The current problem, the accepted reasoning prefix, and a candidate wrong step are sent to the LLM, which is asked to provide local guidance. Conditioned on this guidance, the SLM regenerates the current step. By comparing the SLM's internal representations before and after guidance, RLC extracts a latent correction vector that captures the induced change in reasoning. It retains only effective experiences whose corrected reasoning leads to the correct answer. Each retained experience is stored as a key–value pair, with the reasoning state immediately before the wrong generation as the key and the corresponding correction vector as the value, yielding an experience bank that can be queried in subsequent inference.

During online experience reuse, inference is performed entirely by the SLM, with no further access to the LLM. Before each reasoning step, the SLM uses its current hidden state to retrieve relevant experiences from the bank. The retrieved correction is then applied to subsequent generation through a hidden-state intervention. Besides, not every problem requires assistance from external experiences. To reduce unnecessary retrieval and intervention, we further introduce a lightweight router: problems that the SLM can solve independently are handled by the original SLM, and the experience bank is activated only when a correction is likely to be needed.

We evaluate RLC on multiple reasoning benchmarks using SLMs of different scales. Experimental results show that, without updating any model parameters or invoking the LLM during online inference, RLC consistently improves the reasoning performance of SLMs. For example, RLC improves the average performance of a 7B SLM by 7.97 percentage points, bringing it on par with a 32B SLM. It also improves the performance of a 14B SLM by 5.93 percentage points, achieving results comparable to those of 72B LLMs on several tasks.

2 Related Work

Collaborative Reasoning with Large and Small Language Models. Prior work on combining large and small language models can be broadly divided into distillation-based methods and inference-time collaboration. Reasoning distillation transfers the behavior of a stronger teacher into a smaller student by supervising chain-of-thought rationales (Magister et al., 2023), LLM-generated rationales as additional supervision (Hsieh et al., 2023), executable programs (Zhu et al., 2024). Later studies further improve distillation by filtering trajectories (Wang et al., 2023; Lei et al., 2025), decomposing tasks (Shridhar et al., 2023), designing curricula (Jiang et al., 2025), or incorporating symbolic and structured supervision (Sun et al., 2024b; Liao et al., 2025), with broad CoT collections and mentor-based supervision providing additional rationale sources (Kim et al., 2023; Lee et al., 2024a). These methods can improve the standalone capability of small models, but adapting to new teacher feedback or new domains typically requires another round of parameter updates.

Inference-time collaboration instead allocates computation between models without fully absorbing the teacher into the student. Query-level systems use cascades (Chen et al., 2024), learned routers (Ong et al., 2025), taxonomy-based selection (Shah & Shridhar, 2025), or calibrated confidence (Zhang et al., 2026) to decide whether a problem should be handled by a small or large model. More fine-grained methods move the decision inside the reasoning trajectory: AdaSwitch assigns reasoning tokens to different agents (Sun et al., 2024a), STEER invokes a large model based on step-level confidence (Lee et al., 2026), GlimpRouter estimates step difficulty from first-token entropy (Zeng et al., 2026), TrigReason detects risky reasoning states (Zhao et al., 2026), and token-level routing consults the cloud model only for selected tokens (She et al., 2025). Another line exchanges guidance rather than only routing computation. SMART asks the large model for targeted hints (Kim et al., 2025), MentorCollab adopts corrective segments under trajectory divergence (Wang et al., 2026), Tandem uses the large model as a strategic coordinator (Fu et al., 2026), LM-Guided CoT studies small-model guidance for black-box large-model reasoning (Lee et al., 2024b), and PyroDash learns control tokens for generation-time handoff (Lyu et al., 2026). These approaches reduce unnecessary large-model computation, but their guidance is usually consumed only by the current instance. In contrast, RLC converts effective offline teacher guidance into persistent latent corrections stored in an external Bank, enabling a frozen small model to reuse prior guidance during future reasoning without online large-model calls or parameter updates.

Inference-Time Activation Steering for Reasoning. Activation steering controls generation by modifying hidden representations during inference. Early methods construct steering directions from contrastive prompt pairs (Turner et al., 2023), multiple positive–negative examples (Rimsky et al., 2024), or in-context demonstrations (Liu et al., 2024), providing a lightweight alternative to fine-tuning. Subsequent work studies more controlled intervention operations, including scaling selected activation dimensions (Stoehr et al., 2024), pseudo-rotating hidden states while preserving norms (Pham & Nguyen, 2024), and constraining intervened states to remain close to the model's original activation distribution (Vogels et al., 2026). Activation steering has also been applied to specific behaviors such as debiasing (Li et al., 2025), chain-of-thought compression (Azizi et al., 2026), and cross-lingual reasoning transfer (Maraia et al., 2026).

Recent reasoning-oriented steering methods make interventions more adaptive. ReBalance adjusts steering based on real-time confidence (Li et al., 2026), PDS retrieves input-matched reasoning prototypes (Kayan & Zhang, 2025), K-CAST estimates intervention coefficients from neighboring examples (Valentino et al., 2026), and RISER composes latent reasoning-skill vectors with a lightweight router (Ye et al., 2026). However, these methods mainly rely on global directions, input-level prototypes, or reusable skills, limiting their ability to match the local states that emerge during multi-step reasoning. In contrast, RLC stores state-conditioned latent corrections from local teacher guidance and retrieves them according to the evolving reasoning state, enabling step-specific intervention and abstention when no relevant experience is found.

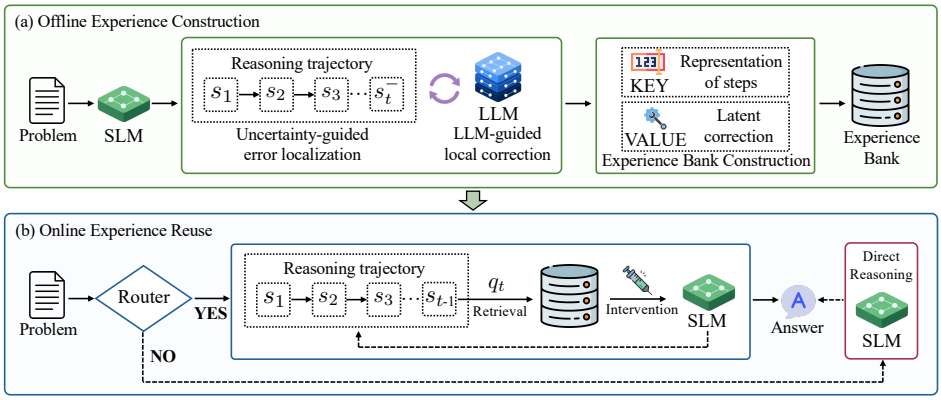


Figure 2: Overall architecture of Reusable Latent Correction (RLC), which consists of offline experience construction and online experience reuse.

3 Problem Formulation

In this paper, we consider a constrained LLM–SLM collaboration setting consisting of a locally deployable small language model (SLM) M_S and a black-box large language model (LLM) M_L. Among them, the parameters and hidden states of M_S are accessible, whereas M_L can only be queried through an API. The LLM is available during offline preparation but cannot be accessed during online inference. Moreover, the parameters of the SLM is fixed throughout the entire process.

Let $\mathcal{D}_{\text{off}} = \{(x_i, a_i)\}_{i=1}^{N}$ denote a set of offline reasoning problems with reference answers. Our objective consists of two stages. During the offline stage, we use natural-language guidance obtained from M_L on $\mathcal{D}_{\text{off}}$ to construct an external experience bank $\mathcal{B}$ without updating the parameters of M_S. During online inference, given a problem $x \in \mathcal{D}_{\text{on}}$, the SLM retrieves relevant corrections from $\mathcal{B}$ according to its evolving reasoning states and uses them to guide subsequent generation. Therefore, online inference is performed solely by M_S with the offline-constructed bank $\mathcal{B}$, without further access to M_L. The overall process can be formulated as:

$$\mathcal{B} = \mathcal{A}_{\text{off}}\left(M_S, M_L, \mathcal{D}_{\text{off}}\right), \quad \widehat{a} = \mathcal{A}_{\text{on}}\left(M_S, x, \mathcal{B}\right), \tag{1}$$

where $\mathcal{A}_{\text{off}}$ denotes the offline procedure for constructing the experience bank, $\mathcal{A}_{\text{on}}$ is the online inference procedure for reusing the stored experiences.

4 Reusable Latent Correction

As shown in Figure 2, we propose **Reusable Latent Correction (RLC)**, which converts one-off LLM guidance into reusable latent corrections for a fixed SLM. During **offline construction**, uncertain reasoning steps are corrected with LLM guidance and stored as KEY–VALUE experiences. During **online reuse**, the SLM retrieves and injects relevant corrections based on its current reasoning state, enabling improved reasoning without online LLM access.

4.1 Offline Experience Construction

Given a problem $x \in \mathcal{D}_{\text{off}}$, the SLM generates a reasoning trajectory $s = (s_1, \ldots, s_T)$ step by step, where each step is separated by a predefined delimiter. Before generating step t, the current context is $c_t = (x, s_{1:t-1})$, and the next step is generated as:

$$s_t \sim M_S\left(\cdot \mid c_t\right), \qquad c_t = (x, s_{1:t-1}). \tag{2}$$

The offline stage then identifies uncertain steps, obtains local LLM guidance, and converts successful corrections into reusable latent experiences.

Uncertainty-guided error localization. Given the reasoning trajectory s, we aim to identify steps at which the SLM has not formed a stable prediction. Since M_S consists of multiple Transformer layers, the representation and prediction of each token are progressively refined from intermediate layers to the final layer. For a confident reasoning step, predictions from intermediate layers are expected to be consistent with the final prediction, whereas large disagreement indicates greater uncertainty (Chuang et al., 2024; Lin et al., 2025). We therefore estimate step-level uncertainty by measuring the divergence between selected intermediate-layer predictions and the final-layer

prediction. Let P_t denote the token positions belonging to step s_t, and let h_τ^l denote the hidden state at token position τ in layer l. For each selected intermediate layer $l \in \mathcal{L}$, we project its normalized hidden state into the vocabulary space:

$$p_l^{(\tau)} = \mathrm{softmax}\left(\frac{W_E\,\mathrm{LN}(h_\tau^l)}{T_1}\right), \qquad u_t = \frac{1}{|\mathcal{L}||P_t|}\sum_{l\in\mathcal{L}}\sum_{\tau\in P_t} D_{\mathrm{JS}}\left(p_N^{(\tau)} \parallel p_l^{(\tau)}\right). \tag{3}$$

where W_E is the output projection matrix, $p_N^{(\tau)}$ is the corresponding final-layer distribution, and T_1 is a temperature parameter. A larger u_t indicates stronger disagreement between intermediate- and final-layer predictions. We identify s_t^- as an uncertain reasoning step when $u_t > \tau_{\mathrm{unc}}$, where τ_{unc} is a predefined threshold that determines whether LLM guidance is triggered for the current step.

LLM-guided local correction. When an uncertain step s_t^- is detected, RLC provides the LLM with the problem x and the partial reasoning trajectory $s_{1:t}$, and asks it to diagnose potential issues in the uncertain step and provide targeted natural-language guidance g_t for correcting that step:

$$g_t = M_L\left(x, s_{1:t-1}, s_t^-\right). \tag{4}$$

Then, the returned guidance g_t is appended to the original reasoning context c_t to form the guided input $\widetilde{c}_t$, which is fed into the SLM to regenerate the current step:

$$\widetilde{c}_t = c_t \oplus g_t = (x, s_{1:t-1}, g_t), \qquad s_t^+ \sim M_S\left(\cdot \mid \widetilde{c}_t\right), \tag{5}$$

where $\oplus$ denotes textual concatenation. Once s_t^+ is generated, the guidance g_t is removed, and subsequent reasoning proceeds from the corrected context $c_{t+1} = (x, s_{1:t-1}, s_t^+)$. Then, the same uncertainty estimation and local correction are repeatedly applied to subsequent reasoning steps until the complete reasoning trajectory is generated.

Experience Bank Construction. After obtaining LLM guidance, we aim to preserve its effect on the SLM rather than the guidance text itself. To this end, RLC stores each candidate experience as a KEY–VALUE pair (k_t, v_t). The key k_t records the SLM reasoning state immediately before the uncertain step is generated and serves as the retrieval representation for matching similar future states, while the value v_t captures the change in the SLM hidden space induced by the LLM guidance.

Before describing the construction of the keys and values, we first introduce the hidden-state representations used by RLC. Let ℓ^* denote the layer used for experience retrieval and hidden-state intervention. For any input sequence z, $h_{\mathrm{last}}^{\ell^*}(z)$ denotes the hidden state of its final token at layer ℓ^*. For a reasoning step s_j, we define its mean representation as $\bar{h}^{\ell^*}(s_j) = \frac{1}{|s_j|}\sum_{\tau\in s_j} h_\tau^{\ell^*}$, where $h_\tau^{\ell^*}$ denotes the hidden state of token τ at layer ℓ^*.

(i) Construction of the retrieval KEY. Based on these representations, we construct the retrieval key k_t as follows. For the first reasoning step, the key is the hidden state of the final token in the problem prompt. For a later step, the key is the mean representation of the preceding reasoning step s_{t-1}:

$$k_t = \mathbb{I}[t=1]\, h_{\mathrm{last}}^{\ell^*}(x) + \mathbb{I}[t>1]\, \bar{h}^{\ell^*}(s_{t-1}), \tag{6}$$

where $\mathbb{I}[\cdot]$ denotes the indicator function. The key therefore describes the SLM state immediately before step s_t is generated, without containing information from the uncertain step itself. Because the same state is available during online inference, RLC can retrieve a relevant correction before a similar uncertain step is generated.

(ii) Construction of the VALUE. We use the latent change induced by LLM guidance as the value v_t of each experience. Specifically, the correction vector contains two complementary components. The *process correction* v_t^{proc} captures how adding the guidance changes the generation-boundary state before the SLM regenerates the current step. The *result correction* v_t^{result} captures the representational difference between the original step s_t^- and the guided step s_t^+. They are defined as:

$$v_t^{\mathrm{proc}} = h_{\mathrm{last}}^{\ell^*}\left(\widetilde{c}_t\right) - h_{\mathrm{last}}^{\ell^*}\left(c_t\right), \quad v_t^{\mathrm{result}} = \bar{h}^{\ell^*}\left(s_t^+\right) - \bar{h}^{\ell^*}\left(s_t^-\right), \quad v_t = \left[v_t^{\mathrm{proc}}; v_t^{\mathrm{result}}\right]. \tag{7}$$

Finally, all KEY–VALUE pairs are collected into the experience bank $\mathcal{B} = \{(k_{t_j}, v_{t_j})\}_{j=1}^m$, where m is the number of retained experiences. It is noting that a candidate correction is retained only if continuing the reasoning process from the regenerated step eventually yields the correct answer. This filtering prevents locally plausible but globally ineffective corrections from being stored in the bank.

4.2 ONLINE EXPERIENCE REUSE

Given a problem x, online inference is performed using only the SLM and the offline-constructed experience bank $\mathcal{B}$, without further access to the LLM. The SLM generates a reasoning trajectory $s = (s_1, \ldots, s_T)$ step by step. Before generating each step, RLC constructs a query from the current reasoning state, retrieves and aggregates relevant corrections from $\mathcal{B}$, and then applies the aggregated correction to the subsequent generation. This process consists of two components: state-conditioned retrieval and aggregation, followed by two-part hidden-state intervention.

State-conditioned Retrieval and Aggregation. After completing step s_{t-1} and before generating step s_t, RLC constructs a query q_t using the same representation employed for the offline keys (i.e., $q_t = \mathbb{I}[t = 1]\, h^{\ell^*}_{\text{last}}(x) + \mathbb{I}[t > 1]\, \bar{h}^{\ell^*}(s_{t-1})$). Using the same representation for offline keys and online queries ensures that the current reasoning state can be matched against the stored experiences. For each bank item $(k_i, v_i) \in \mathcal{B}$, we compute its cosine similarity to the current query as:

$$\rho_{t,i} = \cos(q_t, k_i) = \frac{q_t^\top k_i}{\|q_t\|_2 \|k_i\|_2}. \tag{8}$$

Here, $\rho_{t,i}$ measures the relevance of the i-th stored experience to the reasoning state before step s_t. If the highest similarity is below the threshold τ_{sim}, RLC treats the retrieval as unsuccessful, sets $\mathcal{N}_t = \varnothing$, and generates the current step without correction. Otherwise, the indices of the K most similar experiences form the retrieved set, i.e., $\mathcal{N}_t = \text{TopK}_{i \in \{1,\ldots,|\mathcal{B}|\}}(\rho_{t,i})$ when $\max_i \rho_{t,i} \geq \tau_{\text{sim}}$. When $\mathcal{N}_t \neq \varnothing$, RLC aggregates the correction vectors of the retrieved experiences into a single correction applicable to the current reasoning state. Specifically, the normalized similarity weight of experience i is $\omega_{t,i} = \exp(\rho_{t,i}/T_{\text{ret}}) / \sum_{j \in \mathcal{N}_t} \exp(\rho_{t,j}/T_{\text{ret}})$, and the aggregated correction is $\bar{v}_t = \sum_{i \in \mathcal{N}_t} \omega_{t,i} v_i = [\bar{v}_t^{\text{proc}}; \bar{v}_t^{\text{result}}]$. Here, T_{ret} controls the concentration of the weights. The resulting process and result components are then applied at different stages of SLM generation.

Two-part Hidden-state Intervention. The aggregated correction contains two components that intervene at different stages of generating reasoning step s_t. First, after prefilling the current context $(x, s_{1:t-1})$, we inject the process correction into the last-token hidden state at layer ℓ^*:

$$\widetilde{h}^{\ell^*}_{t,\text{last}} = h^{\ell^*}_{t,\text{last}} + \alpha_p \bar{v}_t^{\text{proc}}. \tag{9}$$

This intervention steers the overall reasoning direction before the next step is generated. Then, during autoregressive decoding, we inject the result correction into the hidden state of every token position τ belonging to the generated step:

$$\widetilde{h}^{\ell^*}_{\tau} = h^{\ell^*}_{\tau} + \alpha_r \bar{v}_t^{\text{result}}, \qquad \tau \in P_t. \tag{10}$$

where P_t denotes the set of token positions in s_t, and α_p and α_r control the strengths of the process and result interventions, respectively.

The process correction shifts the initial direction of the upcoming reasoning step, while the result correction continuously guides its token-level generation. RLC repeats retrieval and intervention before each subsequent step until the SLM produces the final answer. Because retrieval is conditioned on the state preceding a potential error, historical LLM guidance can be reused proactively without online LLM access.

Problem-level Router. In practice, bank augmentation is not always necessary: the original SLM can solve some problems independently, while irrelevant retrieved experiences may introduce unnecessary corrections. We therefore employ a lightweight problem-level router R_ϕ to determine whether the experience bank should be activated based solely on the input problem x, i.e., $r = \mathbb{I}[R_\phi(x) \geq \tau_R]$, where τ_R is the routing threshold. When $r = 0$, the original SLM directly solves the problem. When $r = 1$, step-level experience retrieval and hidden-state intervention are activated. The construction and training details of the router are provided in Appendix B.5.

Besides, we evaluate whether the Router can effectively determine when corrective intervention is needed, with detailed results reported in Appendix C.3.

The complete procedure of RLC is summarized in Algorithm 1 in Appendix E.

5 EXPERIMENTS

To comprehensively evaluate RLC, we investigate the following seven research questions: **RQ1:** How effectively does RLC improve the reasoning performance of SLMs across different model scales? **RQ2:** Can latent corrections constructed from one model scale generalize to models of different scales? **RQ3:** Can reusable latent corrections provide comparable effectiveness to directly obtaining LLM guidance during inference? **RQ4:** Does RLC preserve the inference efficiency of the underlying SLM while improving its reasoning performance?

5.1 BENCHMARKS AND EXPERIMENTAL SETUPS

Benchmarks. We evaluate RLC on five reasoning benchmarks: MATH500 (Lightman et al., 2023), GSM8K (Cobbe et al., 2021), AMC24, AIME (Hendrycks et al., 2021), and GPQA-Diamond (Rein et al., 2024). For offline Bank construction, we use the full GSM8K and MATH training sets, totaling 14,973 problems. These examples are used to generate reasoning trajectories, identify errors, and extract latent corrections from localized LLM guidance. No test examples are used during Bank construction, ensuring separation from downstream evaluation. Details are shown in Appendix A.

Baselines. We compare RLC with two categories of baselines. The first is *LLM–SLM collaborative inference*, including STEER (Lee et al., 2026) and GlimpRouter (Zeng et al., 2026), which invoke an LLM during test-time reasoning. The second is *inference-time representation intervention*, including ReBalance (Li et al., 2026) and PDS (Kayan & Zhang, 2025), which steer SLM reasoning by modifying hidden representations using signals derived from the SLM itself. In contrast, RLC uses LLM guidance only offline and reuses the resulting latent corrections during online SLM inference. Detailed baseline descriptions are provided in Appendix B.

Experimental Setups. We use Qwen2.5-3B/7B/14B-Instruct as locally deployed SLMs and Qwen3-Max as the black-box LLM only for offline experience construction. All SLM parameters remain frozen. Qwen2.5-7B/32B/72B Native are included only as standalone reference models and are not used as teachers. Unless otherwise specified, RLC retrieves the top-3 experiences with $\tau_{\text{sim}} = 0.95$, uses $(\alpha_p, \alpha_r) = (0.5, 0.25)$, and applies retrieval and intervention at the fourth Transformer layer from the output. We report **accuracy (Acc.)** as the primary metric, with **Tok.** denoting the average generated tokens per problem. Additional details are provided in Appendix B.

5.2 OVERALL PERFORMANCE (RQ1)

To evaluate the overall effectiveness of RLC, we conduct experiments on five reasoning benchmarks, with results shown in Table 1 and Figure 3. RLC consistently improves SLM reasoning across model scales and substantially narrows the gap to larger models. In particular, RLC (3B) and RLC (7B) slightly outperform the native 7B and 32B models, respectively, while RLC (14B) achieves performance comparable to the native 72B model. Meanwhile, its output length remains close to that of the corresponding Native models, suggesting that the gains mainly come from more effective reasoning correction rather than longer trajectories. Besides, we have the following observations:

Comparison with baselines. Compared with *representation-intervention baselines*, RLC consistently achieves stronger performance across model scales. The key difference is that existing methods mainly derive generic steering signals from the model's own representations, whereas RLC constructs error-specific corrections from actual reasoning failures and offline LLM guidance. Each correction is further tied to its originating reasoning state, enabling state-adaptive intervention as the trajectory evolves. RLC also compares favorably with *LLM–SLM collaborative inference methods*. While these approaches require online LLM participation, RLC reuses offline guidance and performs inference solely with the SLM. This suggests that effective LLM–SLM collaboration does not necessarily require repeated LLM access, as useful corrective information can be stored and reused for similar reasoning states.

Ablation results of two intervention stages. In Table 1, the ablation results confirm that prefill- and decode-time interventions are complementary. Removing either component degrades performance, with the reduction being particularly pronounced for the 7B model, where the full RLC achieves 55.42 average accuracy compared with 51.15 and 50.77 for the two single-stage variants. Prefill intervention adjusts the reasoning direction before generating a new step, while decode intervention maintains the corrective signal during token generation. Their combination therefore provides correction at different stages of the reasoning process rather than duplicating the same effect.

Table 1: Main results across five reasoning benchmarks. Among them, for STEER and GlimpRouter, $A \rightarrow B$ denotes that model A is assisted by model B during inference. Best and second-best results are highlighted in bold and underline, respectively.

Method	MATH500		GSM8K		GPQA		AMC24		AIME25		Average	
	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.
Qwen2.5-3B Native	67.00	607.84	84.69	299.69	30.30	740.65	4.44	967.53	3.33	888.30	37.95	700.80
Qwen2.5-7B Native	**74.20**	**557.86**	**90.83**	290.12	38.89	**586.81**	20.00	1030.87	**10.00**	931.47	46.78	679.43
ReBalance (3B)	70.60	619.74	86.73	290.96	39.45	724.48	22.22	**764.89**	3.33	**870.37**	44.47	**654.09**
PDS (3B)	71.80	636.74	86.58	295.21	33.84	700.39	22.22	963.78	6.67	1304.10	44.22	780.04
STEER (3B→7B)	68.80	626.16	87.64	346.59	32.32	737.54	**24.44**	857.69	**10.00**	872.47	44.64	688.09
GlimpRouter (3B→7B)	68.40	594.96	84.38	281.51	32.32	638.50	**24.44**	853.87	0.00	928.47	41.91	659.46
RLC (3B)	73.00	576.91	89.99	**272.68**	**40.91**	690.92	**24.44**	992.78	**10.00**	921.93	**47.67**	691.04
Prefill only	71.80	571.79	89.99	272.29	40.91	690.01	24.44	1420.96	3.33	1106.17	46.09	812.24
Decode only	69.80	605.78	89.69	273.01	40.92	691.54	20.00	1326.51	3.33	1148.33	44.75	809.03

Method	MATH500		GSM8K		GPQA		AMC24		AIME24		Average	
	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.
Qwen2.5-7B Native	74.20	557.86	90.83	290.12	38.89	586.81	20.00	1030.87	13.33	1398.97	47.45	772.93
Qwen2.5-32B Native	**83.20**	**530.40**	**95.30**	**231.24**	42.93	**481.80**	**37.78**	947.33	13.33	**839.73**	54.51	**606.10**
ReBalance (7B)	79.20	592.87	91.89	316.16	40.40	728.55	28.89	894.62	13.33	1002.87	50.74	707.01
PDS (7B)	76.80	597.08	91.28	305.93	40.91	703.36	26.67	**791.38**	16.67	1259.30	50.47	731.41
STEER (7B→32B)	77.00	596.10	91.21	309.08	40.91	748.68	22.22	887.07	10.00	863.27	48.27	680.84
GlimpRouter (7B→32B)	77.40	589.24	90.90	270.03	41.92	665.14	26.67	987.84	**20.00**	956.40	51.38	693.73
RLC (7B)	79.80	564.70	94.09	279.07	**45.45**	654.44	**37.78**	1100.38	**20.00**	1300.53	**55.42**	779.82
Prefill only	76.80	577.16	94.01	278.70	44.95	653.01	26.67	914.67	13.33	1329.37	51.15	750.58
Decode only	75.40	577.25	94.09	278.82	45.45	649.17	28.89	1143.00	10.00	1187.13	50.77	767.07

Method	MATH500		GSM8K		GPQA		AMC24		AIME24		Average	
	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.
Qwen2.5-14B Native	77.00	602.48	94.47	290.74	42.93	**421.87**	31.11	766.82	13.33	1122.87	51.77	**640.96**
Qwen2.5-72B Native	**83.80**	567.37	**95.75**	284.14	**52.02**	618.53	**40.00**	865.44	16.67	1169.13	57.65	700.92
ReBalance (14B)	79.40	624.00	94.92	320.64	46.46	651.65	35.56	1230.49	10.00	1440.00	53.27	853.36
PDS (14B)	81.00	602.52	94.92	310.04	47.98	703.52	**40.00**	**762.51**	16.67	1031.60	56.11	682.04
STEER (14B→72B)	79.80	627.11	95.22	342.72	45.96	798.60	31.11	886.20	10.00	**1025.50**	52.42	736.03
GlimpRouter (14B→72B)	77.60	**567.27**	94.84	**260.38**	45.45	653.32	26.67	830.44	13.33	1158.23	51.58	693.93
RLC (14B)	83.00	583.87	95.68	280.45	**52.02**	467.12	37.78	864.58	**20.00**	1120.07	**57.70**	663.22
Prefill only	79.00	577.29	95.38	280.43	52.02	470.86	33.33	1128.38	20.00	1172.20	55.95	725.83
Decode only	78.40	606.58	95.60	281.16	51.01	467.83	33.33	972.69	13.33	1163.13	54.33	698.28

Intervention behavior across tasks. Figure 3 shows that intervention frequency exhibits a clear task-dependent pattern. Competition-level mathematics benchmarks such as AMC24 and AIME require the most frequent intervention, averaging 10.61 and 8.53 interventions, respectively, while GPQA requires only 1.06. MATH500 and GSM8K lie between these two extremes. This pattern suggests that tasks involving longer and more complex reasoning trajectories tend to require more persistent correction, whereas shorter or more localized reasoning processes can often be corrected with fewer interventions. Detailed intervention results are shown in Appendix C.5.

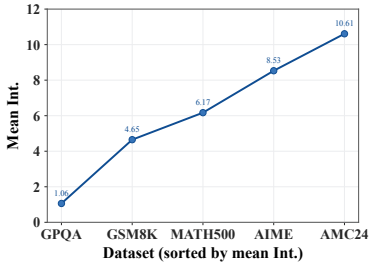


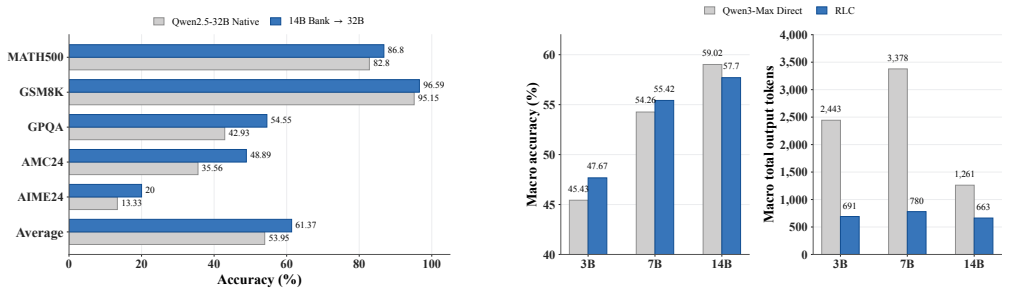
Figure 3: Intervention behavior across tasks. Int. denotes the average number of interventions among Bank-routed problems, averaged across the RLC (3B, 7B, and 14B).

5.3 Cross-Scale Transferability of the Bank (RQ2)

To evaluate whether the Bank can transfer across different model scales, we conduct cross-scale transfer experiments by applying a Bank constructed from one SLM to another SLM of a different scale. Figure 4(a) shows the upward-transfer setting, where a Bank built from Qwen2.5-14B is applied to Qwen2.5-32B. As shown in Figure 4(a), applying the 14B Bank to Qwen2.5-32B improves the average accuracy by 7.42 percentage points, with consistent gains across all five benchmarks. This result indicates that the correction information in the Bank is not restricted to the model scale from which it is collected. Although the corrections are extracted from the internal states of the 14B model, they can still provide effective guidance to a stronger 32B model, suggesting that the Bank captures transferable reasoning-error patterns rather than model-specific output behavior. We also evaluate the reverse 32B→14B transfer, with full bidirectional results reported in Appendix C.2.

5.4 Online Latent Correction vs. Online LLM Feedback (RQ3)

To evaluate whether offline latent corrections can replace direct LLM feedback during inference, we compare RLC with a direct-feedback variant that queries Qwen3-Max at each retrieved candidate



(a) Upward cross-scale Bank transfer from Qwen2.5-14B to Qwen2.5-32B

(b) Comparison of online latent correction and direct Qwen3-Max feedback

Figure 4: (a) Upward cross-scale Bank transfer from Qwen2.5-14B to Qwen2.5-32B. The Bank is constructed with the 14B model and applied to the 32B model for inference. (b) Comparison of online latent correction and direct LLM feedback in terms of accuracy and total generated output tokens, where the token count for direct feedback includes both local-model and API outputs.

Table 2: Inference efficiency comparison of RLC with same-scale and larger Native models. Results report the average inference time per problem, with lower values indicating better efficiency.

Target	Configuration	MATH500	GSM8K	GPQA	AMC24	AIME	Average
3B	3B Native	11.83	5.94	13.73	27.12	**16.76**	**15.08**
	3B RLC	**11.33**	**5.44**	**12.92**	29.10	19.70	15.70
	7B Native	13.12	6.65	13.34	**24.34**	33.09	18.11
7B	7B Native	**13.12**	6.65	**13.34**	**24.34**	33.09	**18.11**
	7B RLC	13.49	**6.45**	15.15	26.69	31.56	18.67
	32B Native	16.73	7.34	15.60	29.26	**27.76**	19.34
14B	14B Native	18.90	8.70	**13.59**	**19.02**	27.68	**17.58**
	14B RLC	**18.44**	**8.42**	15.97	21.81	**26.93**	18.31
	72B Native	24.00	13.78	26.61	37.41	49.63	30.29

step under the same router and retrieval trigger. Figure 4(b) reports the average accuracy and total output tokens. RLC achieves comparable or better accuracy at the 3B and 7B scales while reducing total output tokens by over 70%. At the 14B scale, its accuracy is only 1.32 percentage points lower, with nearly half the output tokens. These results indicate that repeated online LLM feedback is not necessary for effective correction, as offline guidance can retain much of its effect after being converted into latent corrections. Full per-dataset results are provided in Appendix C.4.

5.5 Inference Efficiency of RLC (RQ4)

To evaluate the inference overhead introduced by Bank retrieval and latent intervention, we measure the average wall-clock time per problem for RLC, its same-scale Native counterpart, and a larger standalone Native model. Timing covers the full inference process from input preprocessing and tokenization to final decoding. The results are shown in Table 2. Compared with same-scale Native inference, RLC introduces only a small latency overhead, averaging 3.1%–4.2% across the three model scales. The overhead varies across model–dataset pairs, and RLC is even faster than Native in several cases. This is because retrieval and latent intervention add computation, while successful correction can shorten or simplify subsequent reasoning trajectories. Overall, the additional cost of consulting the Bank remains limited. More importantly, RLC preserves the inference advantage of smaller local models. Across the three comparison groups, RLC is faster than the corresponding larger Native model in 13 of 15 model–dataset comparisons, with a maximum latency reduction of 39.5% for 14B RLC over 72B Native. This shows that RLC can approach the reasoning performance of much larger models while retaining substantially lower inference latency.

6 Conclusion

In this paper, we studied how to improve SLM reasoning while keeping model parameters frozen and avoiding online LLM access. We proposed **Reusable Latent Correction (RLC)**, which consisted of offline experience construction and online reuse. Offline, RLC identified uncertain reasoning steps, used LLM guidance to regenerate them, and extracted latent corrections from the resulting hidden-state changes. Effective corrections were stored as key–value pairs, linking pre-error reasoning states to correction vectors. Online, the SLM retrieved relevant corrections from its current hidden state and injected them into subsequent generation. Experiments across multiple reasoning benchmarks and model scales demonstrated the effectiveness of RLC.

AI USE STATEMENT

In this work, we used generative AI tools to construct synthetic MetaMath-style problems for the Router training set, as described in Appendix B.5. These synthetic problems were used only for Router training . We also used generative AI tools to assist with information retrieval and discovery, including identifying potentially relevant literature, to aid and polish the writing of the manuscript, and to write and debug code. The authors manually verified the cited sources, reviewed and revised all AI-assisted text, and inspected and tested the AI-assisted code for correctness. Generative AI tools were not used to formulate or prove mathematical claims, interpret experimental results, or determine the conclusions of this work. The remaining required-disclosure tasks are not applicable to this work. We take full responsibility for the final content of this work, including all text, code, data, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We support reproducibility by providing a detailed description of RLC and its evaluation throughout the main paper and appendices. Section 4 describes the offline experience construction and online experience reuse procedures. Section 5.1 and Appendix A document the benchmarks. Appendix B specifies the models, decoding settings, Bank construction process, retrieval and intervention hyperparameters, Router training procedure, and answer-evaluation protocol. The complete model prompts and two-stage procedures are provided in Appendices D and E, respectively. Additional detailed experimental results are reported in the remaining appendices. Together, these materials provide the information required to reproduce the reported experiments and verify the main conclusions of this work.

REFERENCES

Seyedarmin Azizi, Erfan Baghaei Potraghloo, Souvik Kundu, and Massoud Pedram. Activation steering for chain-of-thought compression. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 36676–36687, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1828. URL https://aclanthology.org/2026.findings-acl.1828/.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=cSimKw5p6R.

Yung-Sung Chuang, Yujia Xie, Hongyin Luo, Yoon Kim, James R. Glass, and Pengcheng He. Dola: Decoding by contrasting layers improves factuality in large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Th6NyL07na.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Zichuan Fu, Xian Wu, Guojing Li, Yejing Wang, Yijun Chen, Zhao Zihao, Luo Yixuan, Hanyu Yan, Yefeng Zheng, and Xiangyu Zhao. Tandem: Riding together with large and small language models for efficient reasoning. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 42286–42302, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.2098. URL https://aclanthology.org/2026.findings-acl.2098/.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025. doi: 10.1038/s41586-025-09422-z. URL https://doi.org/10.1038/s41586-025-09422-z.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In J. Vanschoren and S. Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/be83ab3ecd0db773eb2dc1b0a17836a1-Paper-round2.pdf.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 8003–8017, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.507. URL https://aclanthology.org/2023.findings-acl.507/.

Wangyi Jiang, Yaojie Lu, Hongyu Lin, Xianpei Han, and Le Sun. Teach small models to reason by curriculum distillation. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 7412–7422, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.376. URL https://aclanthology.org/2025.emnlp-main.376/.

Ceyhun Efe Kayan and Li Zhang. Prototype-based dynamic steering for large language models, 2025. URL https://arxiv.org/abs/2510.05498.

Seungone Kim, Se Joo, Doyoung Kim, Joel Jang, Seonghyeon Ye, Jamin Shin, and Minjoon Seo. The CoT collection: Improving zero-shot and few-shot learning of language models via chain-of-thought fine-tuning. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 12685–12708, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.782. URL https://aclanthology.org/2023.emnlp-main.782/.

Yujin Kim, Euiin Yi, Minu Kim, Se-Young Yun, and Taehyeon Kim. Guiding reasoning in small language models with llm assistance, 2025. URL https://arxiv.org/abs/2504.09923.

Hojae Lee, Junho Kim, and SangKeun Lee. Mentor-KD: Making small language models better multi-step reasoners. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 17643–17658, Miami, Florida, USA, November 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.977. URL https://aclanthology.org/2024.emnlp-main.977/.

Jooyoung Lee, Fan Yang, Thanh Tran, Qian Hu, Emre Barut, and Kai-Wei Chang. Can small language models help large language models reason better?: LM-guided chain-of-thought. In Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue (eds.), *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 2835–2843, Torino, Italia, May 2024b. ELRA and ICCL. URL https://aclanthology.org/2024.lrec-main.252/.

Sangmook Lee, Dohyung Kim, Hyukhun Koh, Nakyeong Yang, and Kyomin Jung. Confidence-guided stepwise model routing for cost-efficient reasoning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(37):31483–31491, March 2026. ISSN 2159-5399. doi: 10.1609/aaai.v40i37.40413. URL http://dx.doi.org/10.1609/aaai.v40i37.40413.

Zhenyu Lei, Zhen Tan, Song Wang, Yaochen Zhu, Zihan Chen, Yushun Dong, and Jundong Li. Learning from diverse reasoning paths with routing and collaboration. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 2832–2845, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.141. URL https://aclanthology.org/2025.emnlp-main.141/.

Yichen Li, Zhiting Fan, Ruizhe Chen, Xiaotang Gai, Luqi Gong, Yan Zhang, and Zuozhu Liu. FairSteer: Inference time debiasing for LLMs with dynamic activation steering. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 11293–11312, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.589. URL https://aclanthology.org/2025.findings-acl.589/.

Yulin Li, Tengyao Tu, Li Ding, Junjie Wang, Huiling Zhen, Yixin Chen, Yong Li, and Zhuotao Tian. Efficient reasoning with balanced thinking, 2026. URL https://arxiv.org/abs/2603.12372.

Huanxuan Liao, Shizhu He, Yupu Hao, Xiang Li, Yuanzhe Zhang, Jun Zhao, and Kang Liu. SKIntern: Internalizing symbolic knowledge for distilling better CoT capabilities into small language models. In Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (eds.), *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 3203–3221, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.215/.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Zhengkai Lin, Zhihang Fu, Ze Chen, Chao Chen, Liang Xie, Wenxiao Wang, Deng Cai, Zheng Wang, and Jieping Ye. Controlling thinking speed in reasoning models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 78300–78347. Curran Associates, Inc., 2025. doi: 10.52202/085713-2623. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/70d46c12643e8f056a693ed43472dc35-Paper-Conference.pdf.

Sheng Liu, Haotian Ye, Lei Xing, and James Y. Zou. In-context vectors: Making in context learning more effective and controllable through latent space steering. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32287–32307. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/liu24bx.html.

Niqi Lyu, Pengtao Shi, Wei Qiu, Jianlin Zhong, Sicong Xia, Jianyao Ma, and Yicheng Ding. Pyrodash: Cost-efficient token-level small-large language model collaborative inference, 2026. URL https://arxiv.org/abs/2607.20327.

Lucie Charlotte Magister, Jonathan Mallinson, Jakub Adamek, Eric Malmi, and Aliaksei Severyn. Teaching small language models to reason. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1773–1781, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-short.151. URL https://aclanthology.org/2023.acl-short.151/.

Gabriele Maraia, Leonardo Ranaldi, Marco Valentino, and Fabio Massimo Zanzotto. Can activation steering generalize across languages? a study on syllogistic reasoning in language models. In Vera Demberg, Kentaro Inui, and Lluís Marquez (eds.), *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2739–2753, Rabat, Morocco, March 2026. Association for Computational Linguistics. ISBN 979-8-89176-380-7. doi: 10.18653/v1/2026.eacl-long.125. URL https://aclanthology.org/2026.eacl-long.125/.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms with preference data. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=8sSqNntaMr.

OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. URL https://arxiv.org/abs/2303.08774.

Van-Cuong Pham and Thien Huu Nguyen. Householder pseudo-rotation: A novel approach to activation editing in LLMs with direction-magnitude perspective. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 13737–13751, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.761. URL https://aclanthology.org/2024.emnlp-main.761/.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Nina Rimsky, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Turner. Steering llama 2 via contrastive activation addition. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15504–15522, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.828. URL https://aclanthology.org/2024.acl-long.828/.

Soham Shah and Kumar Shridhar. Select-then-route : Taxonomy guided routing for LLMs. In Saloni Potdar, Lina Rojas-Barahona, and Sebastien Montella (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 425–441, Suzhou (China), November 2025. Association for Computational Linguistics. ISBN 979-8-89176-333-3. doi: 10.18653/v1/2025.emnlp-industry.28. URL https://aclanthology.org/2025.emnlp-industry.28/.

Jianshu She, Wenhao Zheng, Zhengzhong Liu, Hongyi Wang, Eric P. Xing, Huaxiu Yao, and Qirong Ho. Token level routing inference system for edge devices. In Pushkar Mishra, Smaranda Muresan, and Tao Yu (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 159–166, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-253-4. doi: 10.18653/v1/2025.acl-demo.16. URL https://aclanthology.org/2025.acl-demo.16/.

Kumar Shridhar, Alessandro Stolfo, and Mrinmaya Sachan. Distilling reasoning capabilities into smaller language models. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 7059–7073, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.441. URL https://aclanthology.org/2023.findings-acl.441/.

Niklas Stoehr, Kevin Du, Vésteinn Snæbjarnarson, Robert West, Ryan Cotterell, and Aaron Schein. Activation scaling for steering and interpreting language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 8189–8200, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.479. URL https://aclanthology.org/2024.findings-emnlp.479/.

Hao Sun, Jiayi Wu, Hengyi Cai, Xiaochi Wei, Yue Feng, Bo Wang, Shuaiqiang Wang, Yan Zhang, and Dawei Yin. AdaSwitch: Adaptive switching between small and large agents for effective cloud-local collaborative learning. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8052–8062, Miami, Florida, USA, November 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.458. URL https://aclanthology.org/2024.emnlp-main.458/.

Zhihong Sun, Chen Lyu, Bolun Li, Yao Wan, Hongyu Zhang, Ge Li, and Zhi Jin. Enhancing code generation performance of smaller models by distilling the reasoning ability of LLMs. In Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue (eds.), *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 5878–5895, Torino, Italia,

May 2024b. ELRA and ICCL. URL https://aclanthology.org/2024.lrec-main.521/.

Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J. Vazquez, Ulisse Mini, and Monte MacDiarmid. Steering language models with activation engineering, 2023. URL https://arxiv.org/abs/2308.10248.

Marco Valentino, Geonhee Kim, Dhairya Dalal, Zhixue Zhao, and André Freitas. Mitigating content effects on reasoning in language models through fine-grained activation steering. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(39):33314–33322, March 2026. ISSN 2159-5399. doi: 10.1609/aaai.v40i39.40617. URL http://dx.doi.org/10.1609/aaai.v40i39.40617.

Arthur Vogels, Benjamin Wong, Yann Choho, Annabelle Blangero, and Milan Bhan. In-distribution steering: Balancing control and coherence in language model generation. In Stelios Piperidis, Núria Bel, Henk van den Heuvel, Nancy Ide, Simon Krek, and Antonio Toral (eds.), *Proceedings of the Fifteenth Language Resources and Evaluation Conference*, pp. 2076–2089, Palma de Mallorca, Spain, May 2026. ELRA Language Resource Association. doi: 10.63317/4629fxavjicu. URL https://aclanthology.org/2026.lrec-1.163/.

Haojin Wang, Yike Wang, Shangbin Feng, Hannaneh Hajishirzi, and Yulia Tsvetkov. Mentorcollab: Large-to-small inference-time mentorship for concise reasoning in language models, 2026. URL https://arxiv.org/abs/2602.05307.

Peifeng Wang, Zhengyang Wang, Zheng Li, Yifan Gao, Bing Yin, and Xiang Ren. SCOTT: Self-consistent chain-of-thought distillation. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5546–5558, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.304. URL https://aclanthology.org/2023.acl-long.304/.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement, 2024. URL https://arxiv.org/abs/2409.12122.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Wencheng Ye, Xiaoyang Yuan, Yi Bin, Pengpeng Zeng, Hengyu Jin, Liang Peng, and Heng Tao Shen. Riser: Orchestrating latent reasoning skills for adaptive activation steering, 2026. URL https://arxiv.org/abs/2601.09269.

Wenhao Zeng, Xuteng Zhang, Yuling Shi, Chao Hu, Yuting Chen, Beijun Shen, and Xiaodong Gu. GlimpRouter: Efficient collaborative inference by glimpsing one token of thoughts. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17850–17864, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.885. URL https://aclanthology.org/2026.findings-acl.885/.

Chuang Zhang, Zizhen Zhu, Yihao Wei, Bing Tian, Junyi Liu, Henan Wang, Wang Xavier, and Yaxiao Liu. Confidence-calibrated small-large language model collaboration for cost-efficient reasoning. In Vera Demberg, Kentaro Inui, and Lluís Marquez (eds.), *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume*

1: Long Papers), pp. 4480–4501, Rabat, Morocco, March 2026. Association for Computational Linguistics. ISBN 979-8-89176-380-7. doi: 10.18653/v1/2026.eacl-long.208. URL https://aclanthology.org/2026.eacl-long.208/.

Yi Zhao, Yajuan Peng, Cam-Tu Nguyen, Zuchao Li, Xiaoliang Wang, Xiaoming Fu, and Hai Zhao. TrigReason: Trigger-based collaboration between small and large reasoning models. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 6700–6718, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.333. URL https://aclanthology.org/2026.findings-acl.333/.

Xuekai Zhu, Biqing Qi, Kaiyan Zhang, Xinwei Long, Zhouhan Lin, and Bowen Zhou. PaD: Program-aided distillation can teach small models reasoning better than chain-of-thought fine-tuning. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 2571–2597, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.142. URL https://aclanthology.org/2024.naacl-long.142/.

A Details of Benchmarks

A.1 Evaluation Benchmarks

We evaluate RLC on five reasoning benchmark families covering grade-school arithmetic, general mathematical reasoning, graduate-level scientific question answering, and competition mathematics.

MATH500 (Lightman et al., 2023) [1] is a 500-problem evaluation subset derived from the MATH benchmark. It covers seven mathematical subjects: algebra, counting and probability, geometry, intermediate algebra, number theory, precalculus, and prealgebra, and includes problems from difficulty levels 1 through 5. We use all examples from the `test` split. Each problem requires an open-form mathematical answer.

GSM8K (Cobbe et al., 2021) [2] evaluates multi-step grade-school mathematical reasoning. We use all 1,319 examples from the `main/test` split. Each reference answer contains a natural-language solution followed by the final numerical answer after the `####` delimiter. Only this final numerical answer is used for evaluation.

GPQA-Diamond (Rein et al., 2024) [3] contains graduate-level questions in biology, physics, and chemistry. We use all 198 examples from the `gpqa_diamond/train` split. Although this subset is distributed under a split named `train`, it is used exclusively for evaluation in our experiments. Each question has one correct answer and three incorrect alternatives. Following the standard GPQA evaluation protocol, we shuffle the four alternatives separately for each question using a fixed random seed of 42 and remap the correct option label accordingly.

AMC24 [4] is constructed from the 2024 AMC 12A and AMC 12B problems. The data source removes five problems that require figures: Problems 14, 18, and 22 from AMC 12A and Problems 7 and 19 from AMC 12B, leaving 45 text-only problems. We use all 45 remaining problems and compare the final open-form answer with the provided reference answer.

AIME24 and AIME25 [5] [6] contain the 30 problems from the AIME I and AIME II examinations in 2024 and 2025, respectively. We use the `test` split of each dataset. These benchmarks require short integer answers and emphasize multi-step competition-level reasoning. The 3B experiments use AIME25, whereas the 7B and 14B experiments use AIME24. This choice is kept fixed across all methods compared within each model scale.

[1] https://huggingface.co/datasets/HuggingFaceH4/MATH-500
[2] https://huggingface.co/datasets/openai/gsm8k
[3] https://huggingface.co/datasets/Idavidrein/gpqa
[4] https://huggingface.co/datasets/rawsh/2024_AMC12
[5] https://huggingface.co/datasets/math-ai/aime24
[6] https://huggingface.co/datasets/math-ai/aime25

A.2 Offline Bank Corpus

We first screen the full training splits of GSM8K and MATH, comprising 7,473 and 7,500 problems, respectively, for a total of 14,973 examples. For each model scale, the corresponding local model performs a single greedy Native generation on every problem. Problems answered incorrectly in this pass are retained as a model-specific offline corpus, from which we identify candidate reasoning errors and construct reusable latent corrections. Importantly, the Bank is constructed exclusively from these training examples. No downstream evaluation instance from MATH500, the GSM8K test split, GPQA-Diamond, AMC24, AIME24, or AIME25 is used during Bank construction. The problem-level Router is trained on a separate set of examples, as detailed in Appendix B.5.

B Experimental Details

B.1 Models and Decoding

We use Qwen2.5-3B-Instruct, Qwen2.5-7B-Instruct, and Qwen2.5-14B-Instruct as the locally deployed SLMs. Qwen3-Max serves as the black-box LLM during offline experience construction and is not accessed during standard RLC inference. All Qwen2.5 backbone parameters remain frozen throughout Bank construction and downstream evaluation. Qwen2.5-7B, Qwen2.5-32B, and Qwen2.5-72B are additionally reported as standalone Native references. They are used only for comparison and do not act as teachers for RLC. Unless otherwise specified, we use greedy decoding, limit each reasoning step to at most 512 generated tokens, and set the maximum total output length for each problem to 4,096 tokens.

B.2 Comparison Baselines

Detailed baseline descriptions. We compare RLC with four representative baselines from two related lines of work: LLM–SLM collaborative inference and inference-time representation intervention. For all baselines, we keep the benchmark splits, answer extraction, and evaluation protocol consistent with RLC.

STEER (Lee et al., 2026) is a confidence-guided stepwise model-routing method for cost-efficient reasoning. At each reasoning step, the smaller model first generates or evaluates the next step, and token-level logit confidence scores are aggregated into a step-level confidence estimate. A calibrated routing rule then decides whether the current step should remain with the smaller model or be routed to a larger model. In our comparison, STEER represents online LLM–SLM collaboration: for a backbone SLM of scale A, STEER can invoke a stronger model of scale B during inference, denoted as $A \rightarrow B$ in Table 1. Unlike RLC, STEER does not construct a reusable experience bank and still requires access to the larger model at test time.

GlimpRouter (Zeng et al., 2026) is a training-free stepwise collaborative-inference method that routes reasoning steps by "glimpsing" the first token of the next thought. Instead of generating an entire candidate step before deciding whether to switch models, the smaller model only predicts the initial token distribution of the next step. The entropy of this first-token distribution is used as a lightweight estimate of step difficulty: low-entropy steps are handled by the smaller model, whereas high-entropy steps are delegated to the larger model. As with STEER, we report GlimpRouter under the $A \rightarrow B$ setting. This baseline reduces unnecessary large-model calls compared with full-step verification, but it still depends on online access to a larger model during inference.

ReBalance (Li et al., 2026) is a training-free inference-time steering method designed to balance overthinking and underthinking in reasoning models. It uses confidence-related signals to identify reasoning modes and constructs steering directions from hidden-state prototypes associated with different thinking behaviors. During generation, ReBalance dynamically adjusts the strength and direction of the steering vector according to the current confidence state, aiming to suppress redundant reasoning when the model overthinks and encourage further exploration when it underthinks. In our experiments, ReBalance is applied directly to the same frozen SLM backbones without invoking an external LLM. Compared with RLC, ReBalance relies on global behavior-level steering directions rather than retrieving error-specific corrections conditioned on the current local reasoning state.

PDS (Kayan & Zhang, 2025) is an inference-time representation-intervention method that constructs input-adaptive steering vectors from reasoning prototypes. It first collects activation differences between Chain-of-Thought and neutral prompting conditions, clusters these differences into a set of reasoning prototypes, and then projects the current input representation onto the prototype set to obtain an instance-specific steering vector. This vector is injected into the model hidden states during inference to enhance latent reasoning without changing model weights or prompts. In our comparison, PDS serves as a prototype-based steering baseline. Unlike RLC, which stores key–value corrections derived from concrete local errors and LLM-guided repairs, PDS relies on global prototype-level steering constructed from input-level representation matching and does not explicitly associate a correction with a specific reasoning state.

B.3 Offline Bank Construction

During offline construction, we estimate step-level uncertainty using three output-relative Transformer layers, namely -4, -8, and -12. The hidden states from these layers are projected through the model's output head and compared against the final-layer predictive distribution using Jensen–Shannon divergence. For each candidate reasoning step, we consider the final token of the preceding prefix together with all tokens in the current step. If the resulting sequence contains more than 32 positions, we uniformly subsample at most 32 positions using a fixed stride. The uncertainty score is then computed as the mean divergence over the sampled positions and the three intermediate layers. A reasoning step is identified as a candidate error if its uncertainty score exceeds a predefined construction threshold. For each detected candidate error, we provide Qwen3-Max with the original problem, the accepted reasoning prefix, and the candidate step. Qwen3-Max then produces a localized diagnosis of the current error together with a concrete instruction for regenerating only that step. This guidance is appended to the SLM context as a private correction note and is never exposed as part of the visible reasoning trajectory.

For each successfully repaired step, we construct a key from the reasoning state immediately preceding the detected error. The corresponding value concatenates two complementary correction signals: a *process correction*, defined as the difference between the unguided and guidance-conditioned prefix states, and a *result correction*, defined as the difference between the regenerated and original representations of the current step. A candidate key–value pair is added to the Bank only if the complete corrected trajectory ultimately produces the correct Gold answer. Throughout Bank construction, retrieval, and intervention, all representations are extracted from the fourth Transformer layer counted backward from the output side.

B.4 Online Retrieval and Intervention

For Bank-routed problems, the SLM generates one visible reasoning step at a time. After each step, its current hidden state is used to retrieve the most similar keys. The default similarity threshold is 0.95, with the selected validation values lying between 0.94 and 0.96. We retrieve the top-k experiences and aggregate their values using similarity-based weights; $k = 3$ is used by default and is reduced only when selected on the validation set. All retrieval settings are fixed before test evaluation. The process and result corrections are controlled by α_p and α_r, respectively. We use $(\alpha_p, \alpha_r) = (0.5, 0.25)$ as the default configuration. A weaker setting of $(0.2, 0.05)$ is used for GSM8K and MATH500 to avoid over-correcting trajectories that are often already reliable, whereas $(0.7, 0.3)$ is used for AMC24 and AIME, whose longer competition-level trajectories generally require stronger guidance. The process correction is applied to the final prefix token during prefill, and the result correction is applied to each newly decoded token of the next step.

B.5 Problem-Level Router Training

The problem-level Router is implemented as a binary sequence classifier initialized from Qwen2.5-0.5B-Instruct. It receives only the raw problem text, with a maximum input length of 768 tokens, generated trajectories, retrieved Bank entries, and Gold answers are not used as Router inputs. The supervision is obtained from disagreements between Native and Bank-augmented inference. Problems for which the Bank corrects a Native failure are labeled *rescued* and assigned to the Bank branch, whereas problems for which the Bank changes a correct Native answer into an incorrect one

Table 3: Effect of state-conditioned correction retrieval. Mean Value replaces each retrieved correction with the global mean of all correction values in the Bank, removing the correspondence between the current reasoning state and its retrieved correction.

Model	Variant	MATH500		GSM8K		GPQA		AMC24		AIME†		Average	
		Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.
3B	**RLC**	**73.00**	576.91	**89.99**	272.68	**40.91**	**690.92**	**24.44**	**992.78**	**10.00**	**921.93**	**47.67**	**691.04**
	Mean Value	70.00	**574.73**	86.28	**272.29**	36.36	723.97	20.00	1003.78	**10.00**	963.77	44.53	707.71
7B	**RLC**	**79.80**	**564.70**	**94.09**	**279.07**	**45.45**	654.44	**37.78**	1100.38	**20.00**	**1300.53**	**55.42**	779.82
	Mean Value	76.80	568.91	93.71	279.48	40.40	**652.57**	35.56	**906.07**	10.00	1335.27	51.29	**748.46**
14B	**RLC**	**83.00**	**583.87**	**95.68**	280.45	**52.02**	**467.12**	**37.78**	**864.58**	**20.00**	1120.07	**57.70**	**663.22**
	Mean Value	78.60	592.06	95.30	**280.34**	48.51	470.31	33.33	941.69	10.00	**1105.83**	53.15	678.05

are labeled *destroyed* and assigned to the Native branch. Cases in which both routes are correct or both are incorrect are not used as Router supervision.

Given a problem x, the Router predicts the probabilities of the two inference branches as

$$p_\phi(c \mid x) = \mathrm{softmax}(f_\phi(x))_c, \qquad c \in \{0, 1\}, \tag{11}$$

where $c = 0$ is the *destroyed* class and selects the Native branch, whereas $c = 1$ is the *rescued* class and selects the Bank branch.

The training set combines sampled MATH and GSM8K training problems, GPQA training problems that do not overlap with GPQA-Diamond, and independently constructed MetaMath-style problems based on existing training examples. We keep the rescued and destroyed classes as balanced as possible and exclude all downstream evaluation instances. Each Router is fine-tuned with the standard cross-entropy objective:

$$\mathcal{L}_{\text{router}} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c \in \{0,1\}} \mathbb{I}(y_i = c) \log p_\phi(c \mid x_i). \tag{12}$$

At inference time, the Router is called once on the input problem and uses a fixed threshold $\tau_R = 0.5$:

$$r(x) = \mathbb{I}[p_\phi(y = 1 \mid x) \geq \tau_R]. \tag{13}$$

When $r(x) = 1$, RLC activates Bank retrieval and hidden-state intervention for the subsequent reasoning trajectory; otherwise, the frozen local model completes Native inference.

B.6 Evaluation

We report Gold-answer accuracy using the official reference answers provided by each benchmark. The evaluator first extracts the final answer from each model output. For MATH500, GSM8K, AMC24, AIME24, and AIME25, the answer contained in the final `\boxed{}` expression is extracted when available. For GSM8K reference answers, the final numerical value following the `####` delimiter is used. The extracted prediction and reference answer are then normalized and compared automatically. The normalization procedure removes commas, surrounding whitespace, units, and common formatting wrappers. A prediction is considered correct if it exactly matches the reference answer, is numerically equivalent to it, or corresponds to a symbolically equivalent mathematical expression. For GPQA-Diamond, the evaluator extracts the final predicted option letter and compares it directly with the remapped Gold option letter.

C More Experiments

C.1 Hyperparameter Sensitivity Analysis

C.1.1 Effectiveness of State-conditioned Correction Retrieval

To evaluate whether the effectiveness of RLC depends on matching the current reasoning state with an appropriate correction, we compare the full RLC with a *Mean Value* variant. Mean Value keeps

the same Router and retrieval schedule, but replaces the retrieved correction with the global mean of all Bank values, thereby removing the instance-specific correspondence between keys and corrections. As shown in Table 3, replacing the retrieved correction with the global mean consistently degrades performance. Across different model scales, the average accuracy drops by 3.14–4.55 percentage points, and RLC outperforms Mean Value in 14 of the 15 model–dataset comparisons. Since the Router and retrieval positions remain unchanged, this gap mainly reflects the loss of state-specific correction rather than differences in when intervention is applied. This indicates that the benefit of RLC does not come from simply injecting a generic steering direction. Instead, the retrieved correction needs to correspond to the current reasoning state.

Table 4: Sensitivity of RLC to the injection layer on Qwen2.5-3B.

Injection layer	Acc.
-2	73.00
-4	**75.00**
-8	74.00
-12	74.00
-18	73.00

C.1.2 Sensitivity to the Injection Layer

To evaluate the sensitivity of RLC to the injection layer, we conduct experiments on Qwen2.5-3B using five output-relative Transformer layers: $\{-2, -4, -8, -12, -18\}$. For all candidate layers, the corresponding Banks are constructed from the same reasoning trajectories and teacher feedback, differing only in the hidden states used to form the keys and values. During evaluation, each Bank is applied to its corresponding layer, while the Router, retrieval trigger, top-K setting, and all other inference configurations remain unchanged. For each trial, we randomly sample 70 problems from the MATH training split and 30 problems from the GSM8K training split, forming a 100-example in-domain development set.

The results in Table 4 show that RLC is relatively insensitive to the exact injection layer. The default -4th layer achieves the highest mean accuracy of 75.00%, while the gap between the best and worst candidate layers is only 2.00 percentage points. Since all layer-specific Banks are constructed from identical trajectories and teacher feedback, the observed differences can be attributed primarily to the injection location rather than changes in correction content. This indicates that the stored corrections remain effective across a range of intermediate and late Transformer layers. We further observe that intermediate layers generally perform slightly better than either very early or output-proximal layers. A possible explanation is that earlier layers may provide less abstract reasoning representations, whereas layers close to the output leave less computation for the model to integrate the injected correction. Intermediate layers therefore offer a better balance between representational abstraction and downstream propagation. Based on these results, we use the -4th layer as the default injection layer in the remaining experiments.

C.2 Full Cross-Scale Transfer Results

To evaluate whether the experience Bank can transfer across different model scales, we conduct cross-scale transfer experiments by applying a Bank constructed from one SLM to another SLM of a different scale. Table 5 reports the bidirectional cross-scale transfer results between Qwen2.5-14B and Qwen2.5-32B. In each direction, the experience Bank is constructed from one model during the offline stage and then applied to the other model for online inference.

As shown in Table 5, transferring the Bank constructed with Qwen2.5-14B to Qwen2.5-32B improves the average accuracy by 7.42 percentage points, with gains observed on all five benchmarks. This result provides additional evidence that the latent corrections stored in the Bank can generalize beyond the model used for Bank construction. Despite being derived from the internal representations of Qwen2.5-14B, these corrections remain effective when applied to Qwen2.5-32B, demonstrating a certain degree of cross-scale transferability of the learned corrective experience. Besides, we further observe a clear asymmetry in cross-scale transfer. While the 14B Bank provides a 7.42-point improvement to the 32B model, transferring the 32B Bank back to the 14B model yields only

Table 5: Full cross-scale Bank transfer results.

Method	MATH500			GSM8K			GPQA			AMC24			AIME24			Average	
	Acc.	Tok.	Int.	Acc.	Tok.	Int.	Acc.	Tok.	Int.	Acc.	Tok.	Int.	Acc.	Tok.	Int.	Acc.	Tok.
32B Bank → 14B																	
Qwen2.5-14B Native	76.20	602.48	–	94.47	290.74	–	42.93	634.18	–	31.11	766.82	–	13.33	1122.87	–	51.61	683.42
32B Bank → 14B	**78.40**	**561.40**	3.16	**95.45**	**280.93**	0.37	42.93	647.59	0.32	**33.33**	1138.64	11.74	13.33	1363.53	16.30	**52.69**	798.42
14B Bank → 32B																	
Qwen2.5-32B Native	82.80	530.40	–	95.15	231.24	–	42.93	**481.80**	–	35.56	947.33	–	13.33	**839.73**	–	53.95	**606.10**
14B Bank → 32B	**86.80**	**507.06**	3.03	**96.59**	**229.13**	2.13	**54.55**	512.77	0.70	**48.89**	**908.02**	4.20	**20.00**	960.67	3.39	**61.37**	623.53

Table 6: Effectiveness of the Router across different model scales and reasoning benchmarks.

Route	MATH500		GSM8K		GPQA		AMC24		AIME25		Avg. Acc.
	Acc.	Bank (%)	Acc.	Bank (%)	Acc.	Bank (%)	Acc.	Bank (%)	Acc.	Bank (%)	
Qwen2.5-3B-Instruct											
Native	67.00	0.00	84.69	0.00	30.30	0.00	4.44	0.00	3.33	0.00	37.95
Bank	65.00	100.00	83.70	100.00	31.82	100.00	22.22	100.00	6.67	100.00	41.88
Router	**73.00**	50.60	**89.99**	40.71	**40.91**	48.99	**24.44**	60.00	**10.00**	73.33	**47.67**

Route	MATH500		GSM8K		GPQA		AMC24		AIME24		Avg. Acc.
	Acc.	Bank (%)	Acc.	Bank (%)	Acc.	Bank (%)	Acc.	Bank (%)	Acc.	Bank (%)	
Qwen2.5-7B-Instruct											
Native	74.20	0.00	90.83	0.00	38.89	0.00	20.00	0.00	13.33	0.00	47.45
Bank	74.00	100.00	90.75	100.00	37.88	100.00	35.56	100.00	16.67	100.00	50.97
Router	**79.80**	50.40	**94.09**	32.30	**45.45**	58.08	**37.78**	71.11	**20.00**	76.67	**55.42**

Route	MATH500		GSM8K		GPQA		AMC24		AIME24		Avg. Acc.
	Acc.	Bank (%)	Acc.	Bank (%)	Acc.	Bank (%)	Acc.	Bank (%)	Acc.	Bank (%)	
Qwen2.5-14B-Instruct											
Native	77.00	0.00	94.47	0.00	42.93	0.00	31.11	0.00	13.33	0.00	51.77
Bank	79.40	100.00	93.25	100.00	41.41	100.00	**37.78**	100.00	16.67	100.00	53.70
Router	**83.00**	50.00	**95.68**	14.48	**52.02**	42.42	**37.78**	77.78	**20.00**	76.67	**57.70**

a 1.08-point gain. One possible reason is that smaller models expose more frequent and diverse reasoning failures during Bank construction, resulting in broader coverage of reusable error patterns. In contrast, the errors of larger models are fewer and more concentrated on difficult cases, making the resulting Bank less comprehensive. Meanwhile, a stronger receiving model has greater capacity to follow a transferred correction and continue the subsequent reasoning process, whereas a smaller model may still fail even when the local reasoning direction is corrected. These results suggest that cross-scale transfer is more effective from a smaller-model Bank to a stronger receiver.

C.3 Effectiveness of the Router

To evaluate whether the Router can effectively determine when Bank-based correction is needed, we compare three inference strategies: *Native*, which always uses the original SLM without accessing the Bank. *Bank*, which always activates Bank-augmented inference, and *Router*, which selectively chooses between the two branches. The results are shown in Table 6. The Router achieves the best accuracy in all 15 model–dataset comparisons, outperforming the better of the two fixed strategies by at least 4.00 percentage points in average accuracy at each model scale. More importantly, the advantage of the Router over always using the Bank reveals that Bank augmentation is not a uniformly beneficial operation. Although the stored corrections are constructed from successful error correction, their effectiveness remains state-dependent at test time: a retrieved correction may be useful when the current reasoning state exhibits a similar error pattern, but unnecessary or mismatched intervention can perturb an otherwise correct trajectory. Therefore, routing every problem through the Bank inevitably applies corrections to instances for which no correction is needed.

Besides, a second observation is that the Router becomes increasingly selective as model capability changes, rather than simply reducing Bank usage uniformly. For example, the Bank rate on GSM8K decreases from 40.71% for the 3B model to only 14.48% for the 14B model, while it remains high on AMC24 and AIME across model scales. This suggests that the Router responds to the residual

Table 7: Comparison between online latent correction and direct LLM feedback using Qwen3-Max. Native is omitted because it does not involve any intervention. A dash indicates that RLC makes no online API calls. *Small* denotes the number of output tokens generated by the SLM, *API* denotes the number of output tokens generated by Qwen3-Max, and *Total* is the sum of the two.

Model	Method	MATH500				GSM8K				GPQA				AMC24				AIME			
		Acc.	Small	API	Total	Acc.	Small	API	Total	Acc.	Small	API	Total	Acc.	Small	API	Total	Acc.	Small	API	Total
3B	Qwen3-Max Direct	70.20	792.68	425.23	1217.91	87.82	297.15	126.13	423.28	36.90	896.59	308.09	1204.68	22.22	1529.93	1312.67	2842.60	10.00	2493.67	4033.10	6526.77
	RLC	73.00	576.91	–	576.91	89.99	272.68	–	272.68	40.91	690.92	–	690.92	24.44	992.78	–	992.78	10.00	921.93	–	921.93
7B	Qwen3-Max Direct	79.60	757.36	556.52	1313.88	92.49	302.68	76.49	379.17	46.97	643.12	52.03	695.15	35.56	2329.78	3437.78	5767.56	16.67	2618.57	6114.10	8732.67
	RLC	79.80	564.70	–	564.70	94.09	279.07	–	279.07	45.45	654.44	–	654.44	37.78	1100.38	–	1100.38	20.00	1300.53	–	1300.53
14B	Qwen3-Max Direct	81.60	668.33	452.76	1121.09	94.92	292.36	61.42	353.78	53.03	481.49	77.26	558.76	42.22	934.42	994.51	1928.93	23.33	1241.53	1100.77	2342.30
	RLC	83.00	583.87	–	583.87	95.68	280.45	–	280.45	52.02	467.12	–	467.12	37.78	864.58	–	864.58	20.00	1120.07	–	1120.07

Table 8: Intervention behavior across tasks. We report the average number of interventions among Bank-routed problems. The rightmost column reports the mean across the five datasets for each model, whereas the bottom row reports the mean across the RLC 3B, 7B, and 14B for each dataset.

Model	MATH500	GSM8K	GPQA	AMC24	AIME	Average
RLC (3B)	5.09	3.25	1.80	12.52	10.73	**6.68**
RLC (7B)	4.27	1.69	0.35	8.84	7.13	**4.46**
RLC (14B)	9.15	9.00	1.04	10.46	7.74	**7.48**
Average	**6.17**	**4.65**	**1.06**	**10.61**	**8.53**	**6.20**

reasoning difficulty of each model–task combination: stronger SLMs rely less on external correction for problems they can already handle reliably, but still frequently activate the Bank on challenging problems where substantial reasoning errors remain.

C.4 Per-Dataset Direct-Intervention Results

Table 7 provides the per-dataset values summarized by Figure 4(b).

C.5 Detailed Intervention Statistics

Table 8 reports the full intervention statistics under Bank routing across the 3B, 7B, and 14B models on all five benchmarks. Beyond the observations discussed in Section 5.2, we also find the intervention frequency does not decrease monotonically as the SLM becomes larger. This indicates that the number of interventions is not determined solely by model capacity.

C.6 Case Studies

We present qualitative case studies to illustrate the two stages of RLC: offline experience construction and online experience reuse. The first case (Figure 5) shows how a localized teacher repair is converted into a reusable key–value latent experience. The remaining three cases (Figures 6, 7, and 8) show how retrieved latent corrections modify test-time reasoning trajectories across GSM8K, MATH500, and AMC24. These examples make the mechanism of RLC concrete: the Bank does not store natural-language feedback, and RLC does not call the LLM during online inference. Instead, it reuses validated hidden-state corrections to steer a frozen SLM at local reasoning states.

Offline Bank construction. Figure 5 illustrates how RLC constructs a Bank entry from a local repair. As shown in Figure 5(a), the Native trajectory first reaches a pre-error reasoning state and then produces an incorrect candidate step s_t^- (e.g., "Since D and E are digits, the only valid case is $k = 0$, so $D + E = 8$."). In Figure 5(b), the black-box LLM provides targeted guidance for this uncertain step, consisting of three components: `[DIAGNOSIS]`, `[NEXT_STEP_GUIDANCE]`, and `[MUST_AVOID]`. Conditioned on this guidance, the frozen SLM regenerates the corresponding corrected step s_t^+, as shown in Figure 5(c). The regenerated trajectory is retained only if its final answer is verified as correct, so that the resulting Bank entry is constructed from a validated local

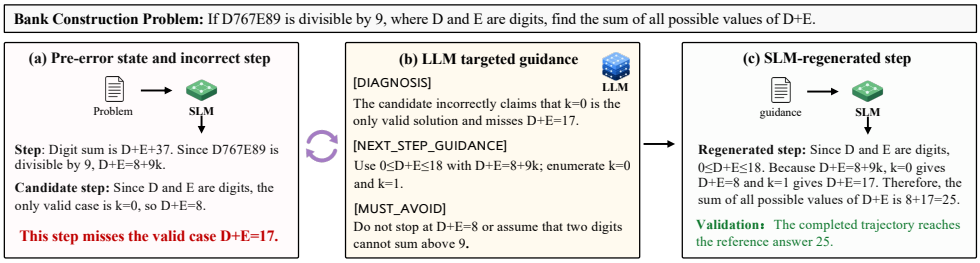


Figure 5: Case study of offline Bank construction in RLC. An incorrect reasoning step is identified, corrected with localized LLM guidance, and converted into a validated reusable latent experience.

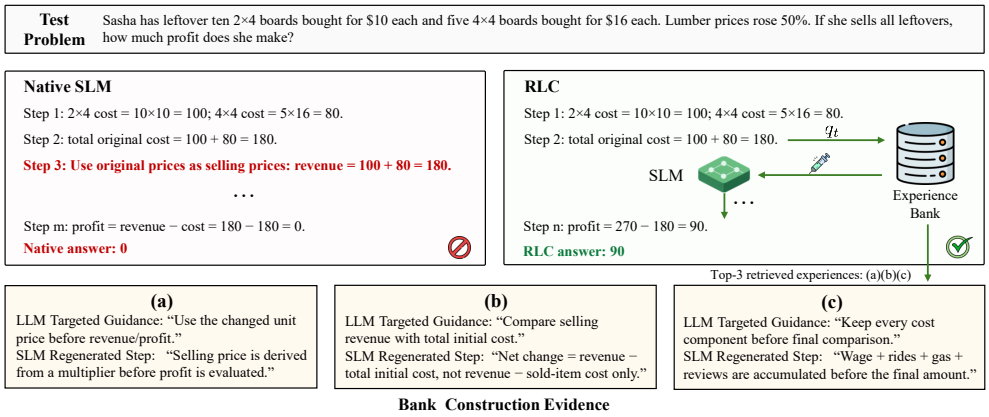


Figure 6: Case study of online latent experience reuse on GSM8K. RLC retrieves relevant latent Bank entries to correct a local reasoning error and steer the frozen SLM toward the correct solution. The displayed Bank Construction Evidence is provided only to interpret the retrieved latent experiences and is not used during online inference.

correction. RLC stores the effect of this repair in hidden-state space: the key represents the reasoning state immediately before the wrong step, while the value contains the process and result correction components induced by the guided regeneration. Thus, the stored experience is not the teacher text itself, but a reusable latent correction associated with a concrete pre-error state.

Online experience reuse. Figure 6 presents a GSM8K example illustrating how RLC corrects a local reasoning error through latent experience reuse. The Native SLM correctly computes the total original cost as $180, but makes an error when estimating the selling revenue: it directly reuses the original prices and obtains a revenue of $180, ignoring the 50% price increase. This local mistake propagates to the final answer, yielding an incorrect profit of zero. At this reasoning state, RLC queries the Bank and retrieves the top-3 relevant latent experiences. The retrieved entries are represented by latent keys and values, rather than natural-language instructions. After their latent values are aggregated and injected into the frozen SLM, the subsequent reasoning trajectory is corrected: the model accounts for the increased selling price, obtains a revenue of $270, and finally produces the correct profit of $90. Since the retrieved Bank entries are latent vectors and cannot be directly visualized, Figure 6 additionally presents their corresponding *Bank Construction Evidence*. For each retrieved latent experience, we show the LLM-provided targeted guidance and the SLM-regenerated reasoning step that were used during offline construction of that entry. These textual elements are shown only to make the retrieved latent experiences interpretable; they are neither stored as the online retrieval content nor reused during test-time inference. Figures 7 and 8 provide two additional examples exhibiting similar local correction behavior on other reasoning tasks.

C.7 Bank Construction Cost and Amortized Reuse

We further analyze the monetary cost of constructing and reusing the RLC Bank. All costs reported in this section refer only to Qwen3-Max API usage and exclude local SLM computation. Based on

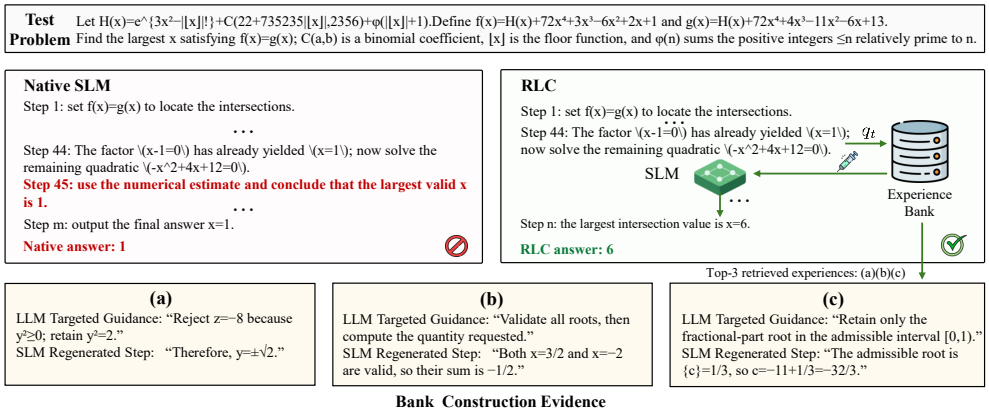


Figure 7: Case study of online latent experience reuse on MATH500. RLC retrieves relevant latent Bank entries to correct a local reasoning error and steer the frozen SLM toward the correct solution. The displayed Bank Construction Evidence is provided only to interpret the retrieved latent experiences and is not used during online inference.

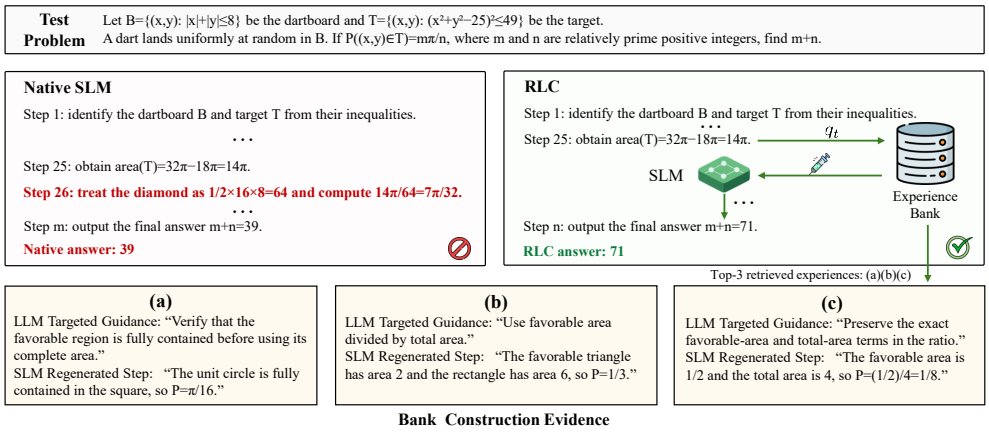


Figure 8: Case study of online latent experience reuse on AMC24. RLC retrieves relevant latent Bank entries to correct a local reasoning error and steer the frozen SLM toward the correct solution. The displayed Bank Construction Evidence is provided only to interpret the retrieved latent experiences and is not used during online inference.

the Qwen3-Max pricing at the time of our experiments[7], we convert the recorded input and output token usage into U.S. dollars using an exchange rate of $1\ \text{USD} = 6.6977\ \text{CNY}$.

Offline Bank construction. Averaged across the Qwen2.5-3B, 7B, and 14B construction processes, each problem requires 8.43 Qwen3-Max calls and $1{,}448.58$ API output tokens on average. Constructing all three Banks incurs a total teacher-API cost of approximately US$52.80.

Repeated online feedback. For comparison, in the direct-feedback evaluation reported in Table 7, each problem receiving direct feedback requires 4.65 Qwen3-Max calls and 738.52 API output tokens on average. A complete direct-feedback evaluation sweep over the three model scales and five benchmarks costs approximately US$7.10.

Amortized reuse. The above comparison highlights an important difference between the two paradigms. Direct feedback incurs additional API cost for every new problem, whereas the constructed Bank can be reused without further teacher-API calls or SLM parameter updates. The one-time Bank construction cost of US$52.80 is equivalent to approximately $52.80/7.10 = 7.44$ complete direct-feedback evaluation sweeps of the scale reported in Table 7. As the Bank is reused across more benchmarks, users, and queries, its amortized teacher-API cost per downstream instance continually decreases.

[7] https://docs.modelstudio.console.alibabacloud.com/zh/model-studio/model-pricing

D MODEL PROMPTS

Stepwise generation instructions used for Bank trajectories

MATH and GSM8K:
You are solving math problems. Separate each reasoning step with a blank line. Put the final answer in \boxed{}.

AMC and AIME:
You are solving math competition problems. Separate each reasoning step with a blank line. Put the final answer in \boxed{}.

GPQA:
You are solving a multiple-choice question. Separate each reasoning step with a blank line. Put the final answer letter in \boxed{}.

Prompt for LLM-guided local correction

```
You are a strict math competition teacher. Silently solve enough of
the problem to judge the current step.
The candidate step may be mathematically wrong, incomplete, or going
down an inefficient path.
Give concrete internal guidance for regenerating ONLY the current next
step. Do not write the whole solution. If the current route is
already valid, preserve it and make only the minimal correction
needed.
Do not give vague advice such as ''consider using inequalities''
without naming the exact inequality, formula, or action.
If the candidate dropped a denominator, changed an equation
incorrectly, made an unjustified assumption, or produced a premature
final answer, explicitly correct that.
If this should not be the final step yet, say what exact operation
should be done next instead of giving a final answer. Avoid changing
to a different global method unless the existing route is truly
invalid.
Original problem:
{question}
Accepted visible solution before the current step:
{accepted_text or [none]}
Current no-thinking candidate:
{current_candidate}
Earlier interventions for this same step:
{history or [none]}
Return only this format:
DIAGNOSIS: one sentence naming the exact issue or saying the step is
acceptable.
NEXT_STEP_GUIDANCE: the concrete formula/calculation/reasoning move the
small model should write for this current step.
MUST_AVOID: one short warning about the main trap.
```

Private correction note for guidance-conditioned regeneration

```
Private correction note:
{guidance}
Continue with only the next visible math step. Do not mention this
note:
```

E ALGORITHM

Algorithm 1: Reusable Latent Correction (RLC)

Input: offline corpus $\mathcal{D}_{\text{off}}$, test problem x, frozen local model f, teacher LLM g, uncertainty threshold τ_u, Router R_ϕ, retrieval threshold τ_{sim}, top-K, and intervention weights (α_p, α_r).
Output: answer $\hat{y}$ and an offline Experience Bank $\mathcal{B}$.

Offline: Experience Bank construction

1. Initialize $\mathcal{B} \leftarrow \varnothing$.
2. **For each** $(x, y) \in \mathcal{D}_{\text{off}}$, generate a stepwise Native trajectory with f and maintain an accepted prefix.
3. At each step t, compute uncertainty u_t. If $u_t \leq \tau_u$, accept the Native step and continue.
4. Otherwise, call g with the problem, accepted prefix, and candidate step to obtain localized guidance; privately condition f on this guidance and regenerate only step t.
5. Extract key k_t from the pre-error reasoning state. Construct v_t^{proc} from the unguided/guided prefix-state difference and v_t^{result} from the original/regenerated step-state difference. Add $(k_t, v_t^{\text{proc}}, v_t^{\text{result}})$ to a pending set.
6. After the trajectory ends, commit its pending entries to $\mathcal{B}$ only if the final answer matches y.

Online: State-conditioned retrieval and intervention

1. Evaluate the problem-level route $r = \mathbb{I}[R_\phi(x) \geq 0.5]$. If $r = 0$, return the Native answer generated by f.
2. If $r = 1$, generate with f one visible reasoning step at a time. Before generating each candidate step, form query state q_t from the current reasoning prefix.
3. Compute the highest similarity $\max_i \rho_{t,i}$ between q_t and the Bank keys. If $\max_i \rho_{t,i} < \tau_{\text{sim}}$, accept the unmodified step and continue. Otherwise, retrieve the top-K most similar Bank entries.
4. Aggregate the retrieved values using normalized similarity weights to obtain $(\bar{v}_t^{\text{proc}}, \bar{v}_t^{\text{result}})$.
5. Add $\alpha_p \bar{v}_t^{\text{proc}}$ to the final prefix-token state during prefill, and add $\alpha_r \bar{v}_t^{\text{result}}$ to each newly decoded token state while regenerating the current step.
6. Append the corrected step to the visible reasoning trajectory and repeat until a final answer is produced; return $\hat{y}$.