LEARNING MULTI-STEP REASONING VIA PERSISTENT LATENT STATE PROPAGATION

Yinxi Li[1] **Jiaao Chen**[2] **Fang Wu**[3] **Jiakai Yu**[4] **Heli Qi**[5]
Weihao Xuan[5,6] **Haokai Zhao**[7] **Pengyu Nie**[1] **Di Jin**[2*] **Xiangru Tang**[8*]
[1]University of Waterloo [2]Eigen AI [3]Stanford University
[4]University of Michigan, Ann Arbor [5]RIKEN AIP [6]University of Tokyo
[7]University of New South Wales [8]Yale University

ABSTRACT

Large language models (LLMs) typically rely on chain-of-thought (CoT) prompting for multi-step reasoning. While effective, this paradigm is brittle, data-intensive, and tightly constrained by token-level generation. Recently, latent reasoning has emerged as an alternative approach. Representative methods, such as HRM and TRM, perform complex inference entirely within their hidden, continuous state space. However, existing studies typically remain confined to direct prediction tasks and operate over restricted representational vocabularies. In this paper, we propose a lightweight Step-wise Persistent Latent Reasoner (SPLR) that performs explicit multi-step reasoning while maintaining persistent hidden-state propagation across steps. SPLR outputs step-wise intermediate hypotheses, optionally consumes external observations, and refines a compact latent state via hierarchical latent dynamics instead of growing a long token-level trace. In controlled experiments on GSM8k-Aug where all architectures are trained from scratch, SPLR achieves 79.4% on MultiArith, substantially higher than the GPT-2 baseline (18.9%). This indicates strong robustness under distribution shift with only 89M parameters. The codebase is open-sourced at `https://github.com/larryinx/splr`.

1 INTRODUCTION

Multi-step reasoning is a crucial requirement for LLMs to solve complex problems, such as mathematical problems, tool-augmented question answering, code generation, and planning (Xia et al., 2025; Copet et al., 2025; Wu et al., 2025b;c; Tang et al., 2025a;b; Chen et al., 2025; Meixin et al., 2025; Wang et al., 2026; Yang et al., 2026b). The dominant interface for eliciting such behavior is *chain-of-thought* (CoT) prompting (Wei et al., 2022), where LLMs externalize intermediate reasoning as natural language tokens. While effective, explicit CoT couples "how much the model thinks" to "how many tokens it prints". Consequently, longer CoT requires longer generated traces, which can be costly, brittle, and sensitive to formatting (Lehnert et al., 2024). Moreover, since intermediate steps are expressed in the same output channel as the final answer, errors in early steps (e.g., arithmetic errors) can be accumulated and amplified. Besides, training models to reliably produce well-formed CoT typically demands substantial supervision (Chen et al., 2024).

These issues are especially pronounced in interactive reasoning settings such as ReAct (Yao et al., 2022). In a ReAct loop, the model alternates between producing intermediate *actions* (e.g., a math expression) and consuming *observations* (e.g., the evaluated result). External feedback can prevent error accumulation by anchoring each step to a deterministic evaluator, but the overall reasoning process is still implemented as token generation over an ever-growing textual history. This motivates architectures that can maintain a running internal state of reasoning progress while still producing explicit intermediate outputs when needed.

Recent work, instead, has explored "*thinking in continuous space*", where reasoning is carried via latent representations rather than explicit text (Deng et al., 2024; Hao et al., 2024; Ye et al., 2026).

*Corresponding authors. Emails: `xiangru.tang@yale.edu`, `di@eigenai.com`

Hierarchical latent reasoning architectures such as Hierarchical Reasoning Model (HRM) (Wang et al., 2025) and Tiny Recursive Model (TRM) (Jolicoeur-Martineau, 2025) iterate over internal states to refine predictions. However, these approaches are often studied in regimes that are either tightly coupled to a specific backbone and decoding procedure (latent-token methods), or confined to direct prediction with restricted output interfaces (hierarchical refinement models). These limitations make it unclear how to combine (i) *explicit multi-step intermediate predictions*, (ii) *optional external feedback*, and (iii) *persistent latent computation across steps* in a lightweight, controlled setting.

In this work, we propose the Step-wise Persistent Latent Reasoner (SPLR), a lightweight model that performs explicit multi-step reasoning while maintaining *persistent latent state propagation* across steps. The core idea is to shift explicit reasoning from token-level CoT traces to a sequence of refinement steps. At each step, SPLR produces an intermediate hypothesis via a ReAct action or a full CoT reasoning step, and optionally receives an external observation. Instead of representing all past reasoning as an ever-longer token prefix, SPLR carries forward a compact latent state that summarizes reasoning progress and is continuously refined over steps. Within each step, the latent dynamics are implemented via a hierarchical refinement procedure, with separate low-level and high-level latent components that are iteratively updated before decoding the next intermediate hypothesis. This design preserves the benefits of explicit intermediate supervision and tool interaction, while decoupling multi-step reasoning from explicit token generation.

We train SPLR on multi-step mathematical reasoning using GSM8k-Aug (Deng et al., 2023; 2024), and evaluate its generalization ability on both in-domain (GSM8K (Cobbe et al., 2021)) and out-of-domain benchmarks (GSM-Hard (Gao et al., 2023), MultiArith (Roy & Roth, 2015), SVAMP (Patel et al., 2021)). To isolate the architectural inductive bias from pretraining, we train all models from scratch on the same amount of data and with the same optimization budgets. We compare SPLR against different GPT-2 baselines (Radford et al., 2019), which are trained with direct prediction, reasoning-only, or ReAct supervision, respectively. Experiments show that SPLR enables competitive multi-step reasoning. Particularly, combining SPLR with ReAct yields particularly strong robustness under distribution shift.

The main contributions of this work are threefold:

- We introduce SPLR, a step-wise reasoning architecture that produces explicit intermediate hypotheses while propagating a persistent latent state across steps, and is compatible with both reasoning-only and ReAct supervision.
- We conduct a controlled empirical study of multi-step reasoning on mathematical problems, showing how SPLR can learn effective reasoning strategies from scratch under limited supervision.
- We perform ablation studies to understand the impact of persistent latent state propagation, adaptive halting, and curriculum learning on SPLR's performance and training stability.

2 Step-wise Persistent Latent Reasoner

In this section, we present the Step-wise Persistent Latent Reasoner (SPLR) (Figure 1), which progressively refines intermediate predictions through a sequence of latent computation cycles. Our key idea is to reason at the level of *refinement steps* rather than token-level chain-of-thought (CoT) generation: each step produces an intermediate hypothesis that is carried forward and refined by subsequent steps while maintaining a continuous hidden-state trace of reasoning progress.

2.1 Overview

We formulate multi-step reasoning as the evolution of a *persistent* latent state across a sequence of refinement steps. Let x_0 denote the original input. At step t, the model produces an intermediate prediction $\hat{y}_t$ and may optionally receive an external feedback signal o_t (e.g., a tool-call result or other deterministic evaluator output). To unify both purely generative and feedback-driven settings, we construct a step-dependent conditioning context

$$c_t \triangleq \Psi\big(x_0, \hat{y}_{t-1}, o_{t-1}\big), \tag{1}$$

where Ψ denotes sequence concatenation. In purely generative settings without external feedback, o_{t-1} is empty at every step.

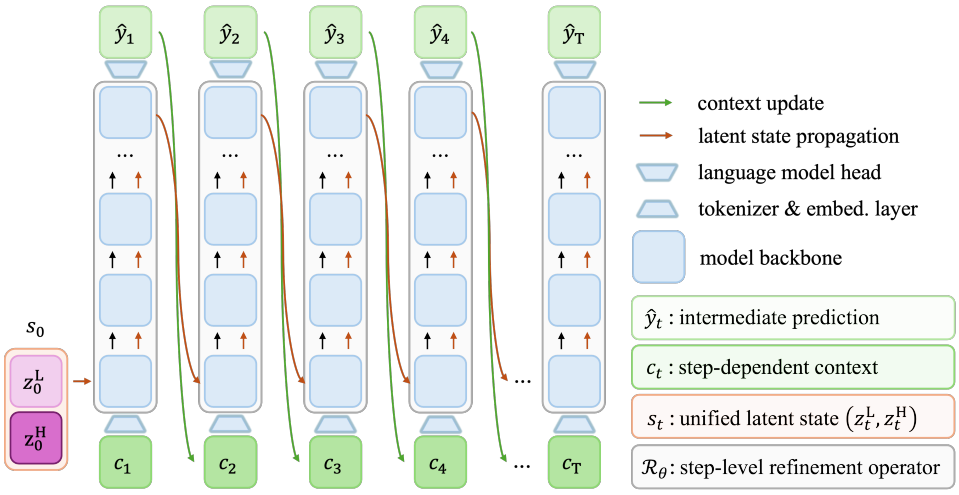

Figure 1: Overview of SPLR for explicit step-wise reasoning. Within each step, SPLR uses a recursive refinement backbone to update a latent state and decode the next intermediate output. Across steps, the refined latent state is propagated forward so that later reasoning steps can build on earlier progress. Appendix D provides backbone details.

Motivated by prior latent reasoning architectures that separate abstract and detailed computation, our model maintains two latent states at each refinement step: a low-level state z_t^L that serves as a persistent planning representation governing the reasoning dynamics, and a high-level state z_t^H that encodes a latent working hypothesis of the current solution. We group them into a unified latent state $s_t \triangleq (z_t^H, z_t^L)$.

At step t, the model updates the latent state s_t conditioned on the current context c_t via a hierarchical refinement procedure and then produces an intermediate prediction $\hat{y}_t$ for the next refinement step. Specifically, we parameterize the refinement dynamics with two modules: a low-level refinement network f_L that updates z_t^L, and a high-level aggregation network f_H that updates z_t^H (details in Sec. 2.2). We then define the step-level refinement operator $\mathcal{R}_\theta$ (parameterized by θ) as the composition of K rounds of alternating low-level updates on z_t^L and high-level updates on z_t^H. Here K is adaptively determined by a halting confidence q_t predicted from the latent state. Formally, we first update the latent state and then decode the intermediate prediction as

$$s_t = \mathcal{R}_\theta(c_t, s_{t-1}), \qquad \hat{y}_t = \texttt{lmhead}(s_t). \tag{2}$$

The operator $\mathcal{R}_\theta$ preserves and refines states across steps, yielding a persistent latent trajectory, while `lmhead` reads out the intermediate prediction from the updated latent state. Let T denote the termination step. The reasoning process terminates when the halting confidence satisfies $q_t \geq \tau$ and the model generates an end-of-sequence token `<eos>` in $\hat{y}_t$. The corresponding prediction $\hat{y}_T$ is returned as the final answer. This design decomposes reasoning into a sequence of adaptive latent refinement steps, while maintaining an internal reasoning trajectory and avoiding the need to externalize intermediate reasoning in natural language.

2.2 Hierarchical Refinement within Each Step

Following the recent hierarchical reasoning architectures (Wang et al., 2025; Jolicoeur-Martineau, 2025), each reasoning step of SPLR is realized through a hierarchical latent refinement procedure with a clear *outer–inner* nesting: within each outer round, we first perform several low-level refinements, and then perform one high-level aggregation. Concretely, within step t we perform K *macro-iterations*. In macro-iteration k, we run J outer rounds, and each outer round consists of I successive low-level updates followed by one high-level update.

Low-level refinement (inner loop). Let $z^H_{t,[k,j]}$ denote the high-level state at the beginning of the j-th outer round in macro-iteration k, and let $z^L_{t,[k,j,0]}$ be the corresponding low-level state. The

inner refinement updates are

$$z^L_{t,[k,j,i+1]} = f_L\big(z^L_{t,[k,j,i]}, z^H_{t,[k,j]}, c_t\big), \quad i = 0, \ldots, I-1. \tag{3}$$

High-level aggregation (outer loop). After the I low-level updates, the high-level state aggregates once:

$$z^H_{t,[k,j+1]} = f_H\big(z^H_{t,[k,j]}, z^L_{t,[k,j,I]}\big), \quad j = 0, \ldots, J-1. \tag{4}$$

We carry the refined low-level state forward by setting $z^L_{t,[k,j+1,0]} = z^L_{t,[k,j,I]}$ for the next outer round. After completing J outer rounds, we start the next macro-iteration from the terminal states via a gradient stop operator $\mathrm{StopGrad}(\cdot)$, which detaches gradients across macro-iterations while preserving forward information flow:

$$z^H_{t,[k+1,0]} = \mathrm{StopGrad}\big(z^H_{t,[k,J]}\big), \quad z^L_{t,[k+1,0,0]} = \mathrm{StopGrad}\big(z^L_{t,[k,J-1,I]}\big). \tag{5}$$

After K macro-iterations, we set $z^H_t = z^H_{t,[K-1,J]}$ and $z^L_t = z^L_{t,[K-1,J-1,I]}$, and group them as $s_t = (z^H_t, z^L_t)$, which is then forwarded to the next reasoning step in Eq. 2.

Adaptive macro-iteration termination. Different inputs may require different amounts of computation *within each step*. We therefore equip the hierarchical refinement procedure with an adaptive halting mechanism over the macro-iterations. For each macro-iteration k, a halting head predicts a confidence score $q_{t,k}$ from the updated high-level state (e.g., $z^H_{t,[k,J]}$):

$$q_{t,k} = \sigma\big(h(z^H_{t,[k,J]})\big). \tag{6}$$

where h is a small MLP and σ is the sigmoid function. In both training and inference, we stop the macro-iterations when $q_{t,k}$ exceeds a fixed threshold τ or when a maximum halt steps limit $K_{\max}$ is reached:

$$K_t = \min\{k : q_{t,k} \geq \tau\} \ \text{(capped by } K_{\max}\text{)}. \tag{7}$$

2.3 Multi-Step Reasoning with Latent State Propagation

The SPLR performs explicit multi-step reasoning by iteratively refining intermediate predictions while propagating a persistent latent state across steps. Unlike chain-of-thought methods that externalize a full reasoning trace, we only output intermediate solutions $\hat{y}_t$ and keep the refinement dynamics in latent space. At each step t, the model updates s_{t-1} under a step-dependent context c_t (Eq. 1) and then reads out the next intermediate prediction $\hat{y}_t$ (Eq. 2). Crucially, long-range reasoning information is carried in the propagated latent state rather than an ever-growing textual history: the latent state s_t provides a compact summary of past reasoning progress, preventing unbounded growth of the conditioning sequence while still enabling iterative refinement.

To achieve this, a key feature is that the latent state is carried forward across steps without re-initialization. Let $z^L_{t,(0)}$ and $z^H_{t,(0)}$ denote the initial states for the within-step refinement procedure at step t. We initialize them from the terminal states of step $t-1$:

$$z^L_{t,(0)} = \mathrm{StopGrad}\big(z^L_{t-1}\big), \quad z^H_{t,(0)} = \mathrm{StopGrad}\big(z^H_{t-1}\big). \tag{8}$$

This yields a single continuous latent trajectory across steps while keeping step-wise training stable.

In inference, we run the refinement process for $t = 1, \ldots, T$ until the model explicitly predicts <eos> token in $\hat{y}_t$, and then return the final prediction $\hat{y}_T$.

2.4 Training Objective

We apply supervision at each refinement stage to encourage progressively improved intermediate solutions. During training, we use *teacher forcing* across steps, i.e., the next-step context is constructed using the ground-truth intermediate target y_{t-1} (and the corresponding feedback o_{t-1} when applicable) rather than the model prediction $\hat{y}_{t-1}$.

Supervision at each macro-iteration. Within each reasoning step t, we attach supervision after each macro-iteration k by reading out an intermediate prediction $\hat{y}_{t,k}$. Let y_t denote the target for step t (with y_T corresponding to the final desired output). The step-wise prediction loss is

$$\mathcal{L}_{t,k}^{\text{pred}} = \ell\big(\hat{y}_{t,k}, y_t\big), \tag{9}$$

where ℓ is the cross-entropy loss. In implementation, we backpropagate and perform an optimizer update after each macro-iteration k (i.e., optimizing online over the refinement trajectory).

Halting loss via correctness supervision. We train the halting head to predict whether the current intermediate prediction is already correct. Let $q_{t,k} \in \{0, 1\}$ be a binary accuracy label computed by an exact-match applied to $\hat{y}_{t,k}$. We minimize the binary cross-entropy between the halting confidence $\hat{q}_{t,k}$ (Eq. 6) and $q_{t,k}$:

$$\mathcal{L}_{t,k}^{\text{halt}} = \lambda \, \text{BCE}\big(\hat{q}_{t,k}, q_{t,k}\big), \tag{10}$$

where λ is a hyperparameter weight on the halting loss term.

The per-macro-iteration training loss is

$$\mathcal{L}_{t,k} = \mathcal{L}_{t,k}^{\text{pred}} + \mathcal{L}_{t,k}^{\text{halt}}. \tag{11}$$

Our overall objective is to minimize $\mathcal{L}_{t,k}$ over all training steps t and macro-iterations k.

3 Experimental Design for Multi-Step Reasoning

In this section, we describe our experimental design for evaluating SPLR on multi-step mathematical reasoning. Inspired by prior latent-reasoning work by Deng et al. (2024); Hao et al. (2024); Shen et al. (2025); Wei et al. (2025), we train on GSM8k-Aug and measure generalization on a suite of math word-problem benchmarks under both in-domain and out-of-domain settings. Beyond reporting benchmark performance, our experimental design uses a controlled training regime to isolate the effects of architectural inductive bias and multi-step supervision. Whereas most existing latent reasoning methods build on large-scale pretrained language models, we study whether reusable reasoning strategies can emerge from limited supervision when training from scratch. Concretely, we train all compared models under matched supervision and optimization budgets, and analyze how different supervision signals and interaction patterns influence reasoning learning and generalization.

3.1 Experimental Setup

Training datasets. We use **GSM8k-Aug** (Deng et al., 2023; 2024) as our primary training corpus for SPLR models. GSM8k-Aug augments the original GSM8k training split (Cobbe et al., 2021) to 385K examples via GPT-4 generated solutions, and removes free-form natural-language rationales, retaining only a sequence of structured mathematical expressions where each step is logically linked to the previous one. Based on GSM8k-Aug, we perform a lightweight normalization to support two training setups: a *ReAct*-style format and a *reasoning-only* format. Full dataset details and the normalization procedure are provided in Appendix B.

Evaluation benchmarks. We report in-domain (ID) performance on the **GSM8k** test set (Cobbe et al., 2021). To evaluate robustness under distribution shift, we further assess mathematical reasoning on three out-of-domain (OOD) benchmarks: (1) **GSM-Hard** (Gao et al., 2023), a modified GSM8k test split where numbers are replaced with larger magnitudes to increase difficulty; (2) **MultiArith** (Roy & Roth, 2015), a subset of MAWPS (Koncel-Kedziorski et al., 2016) consisting of multi-step arithmetic word problems; and (3) **SVAMP** (Patel et al., 2021), a dataset of grade-school arithmetic word problems with simple variations designed to test robustness. Additional details are provided in the Appendix B.

3.2 Model Variants and Training Setup

Baseline settings. To compare hierarchical latent reasoning with standard autoregressive transformers under limited supervision, we train GPT-2 (Radford et al., 2019) models from random initialization on GSM8k-Aug as baselines. All baseline models share the same architectural capacity

and training budget, enabling a controlled comparison focused on reasoning inductive bias rather than pretrained knowledge. We consider three training variants for the baselines: (i) *Direct prediction*, where model is trained only on final answers without intermediate steps; (ii) *Reasoning-only*, where step-level intermediate expressions are provided as supervision; (iii) *ReAct-style*, where each intermediate prediction is paired with an external observation and fed back into subsequent inputs. During evaluation, ReAct-style GPT-2 performs multi-step generation with interleaved observation feedback until termination. More implementation details are provided in the Appendix C.

Training interaction strategies. We consider two training variants in this work, *ReAct-style* (Yao et al., 2022) and *reasoning-only* (Wei et al., 2022), which differ in the form of intermediate supervision and the availability of external feedback during training. Under the ReAct-style variant, each reasoning step is supervised with an explicit action–observation pair: the model predicts an intermediate action $\hat{y}_t$ (i.e., a mathematical expression), and the corresponding observation o_t is provided as part of the training signal. In contrast, under the reasoning-only setup, the model does not receive any external feedback. Each reasoning step is supervised solely by the intermediate target sequence, which includes both the expression and its evaluated result as a single reasoning unit. In both variants, training is performed with teacher forcing across reasoning steps, where the step-dependent context is constructed using the ground-truth intermediate targets and observations rather than the model's own predictions (see Section 2.4).

Training-Time Latent State Ablation. To assess the role of persistent latent trajectories during learning, we further construct an ablated variant in which latent state propagation across reasoning steps is removed during training. Specifically, instead of propagating s_{t-1} to initialize s_t, we re-initialize the latent state at each step. Since this prevents information accumulation in latent space, we compensate for the lack of latent memory by explicitly conditioning each step on all previous intermediate predictions and observations:

$$c'_t \triangleq \Psi(x_0, y_{<t}, o_{<t}). \tag{12}$$

This design reduces the model to a semi-autoregressive reasoning paradigm, where long-range dependencies are maintained through token-level context rather than latent state propagation. It thus provides a controlled comparison between implicit latent memory and explicit sequential conditioning. The corresponding results are reported in Section 4.1 and further analyzed in Section 4.3.

Curriculum Learning. Inspired by iCoT and Coconut (Deng et al., 2024; Hao et al., 2024), we adopt a curriculum learning strategy (Bengio et al., 2009) to facilitate stable training of multi-step reasoning. Specifically, we train all SPLR models in four stages, where the maximum number of reasoning steps is progressively increased to 2, 4, 6, and 12, respectively. This maximum step limit corresponds to the horizontal depth of the refinement process illustrated in Figure 1. To isolate the effect of curriculum learning, we also evaluate an ablated setting in which the maximum number of reasoning steps is fixed to 12 throughout training, while keeping all other training configurations identical. Detailed training schedules are provided in Appendix C.2.

3.3 Evaluation and Inference Setup

Inference-time halting configuration. During inference, we control the maximum number of reasoning steps via the halting mechanism described in Section 2.2. Unless otherwise specified, we set the maximum allowable reasoning steps to $K_{\max} = 2$ for all evaluation experiments. Additionlly, to study the impact of inference-time computation budgets, we further analyze model performance under different values of $K_{\max}$ at evaluation time, while keeping all other settings unchanged. The corresponding results and comparisons are reported in Section 4.1 and full inference hyperparameter details are provided in Appendix C.

Inference-Time Latent Disruption. To isolate the role of latent state propagation during inference, we additionally evaluate a variant in which the latent state is not or partially propagated across reasoning steps at test time. Specifically, while the model is trained with full latent state propagation, we re-initialize the latent state at each inference step, preventing the accumulation of reasoning information in latent space across steps. We focus this analysis on the ReAct-style interaction

Table 1: Benchmark performance of SPLR and Transformer (GPT-2 from scratch) baselines. Best results are highlighted.

Model (Backbone)	Strategy	GSM8K	GSM-Hard	MultiArith	SVAMP
Transformer (GPT-2)	Direct prediction	2.7	0.7	1.1	2.7
Transformer (GPT-2)	Reasoning-only	22.0	4.4	25.0	18.7
Transformer (GPT-2)	ReAct	21.8	4.5	18.9	18.7
SPLR (TRM)	Reasoning-only	19.2	4.5	22.8	22.0
w/o latent prop.	Reasoning-only	22.0	**5.1**	21.1	22.0
w/o curriculum.	Reasoning-only	17.9	4.2	36.7	16.3
SPLR (TRM)	ReAct	**22.6**	4.4	**79.4**	**29.3**
w/o latent prop.	ReAct	21.7	3.9	43.3	29.0
w/o curriculum.	ReAct	17.9	3.6	65.6	24.3

regime, where externally verified observations anchor each reasoning step; this ensures that any performance change under latent disruption can be attributed to the latent state's role in accumulating reasoning progress, rather than being confounded with error propagation in self-generated intermediate computations. The resulting performance differences and qualitative analyses are discussed in Section 4.3.

4 Empirical Results with SPLR

Overall, our experiments are designed to answer the following research questions:

(RQ1) Can hierarchical latent reasoning architectures learn effective multi-step reasoning strategies from scratch under limited supervision?

(RQ2) How do different interaction patterns: reasoning-only versus external feedback (ReAct-style), affect the model's reasoning capability and generalization?

(RQ3) What is the role of persistent latent state propagation across reasoning steps in enabling effective multi-step reasoning?

(RQ4) Does curriculum learning improve the model's ability to learn complex reasoning strategies?

4.1 Main Results

Table 1 summarizes the main benchmark results, addressing **RQ1**: whether hierarchical latent reasoning architectures can learn effective multi-step reasoning strategies *from scratch* under limited supervision. All models are trained without pretrained backbone weights under matched data and optimization budgets. Specifically, we evaluate the SPLR models checkpoint at 260k training steps, whereas GPT-2 baselines are trained to convergence under the same data regime (details in Appendix C). Across all benchmarks, SPLR consistently outperforms GPT-2 baselines trained with direct prediction, reasoning-only, or ReAct-style supervision. Notably, **SPLR with ReAct-style interaction** achieves the strongest overall performance, with substantial gains on MultiArith and SVAMP, demonstrating that hierarchical latent reasoning can effectively acquire reusable multi-step reasoning strategies even without large-scale pretraining.

Beyond final accuracy, Figure 2 analyzes inference-time computation control by varying the maximum number of refinement steps $K_{\max}$ and enabling or disabling the halting mechanism. We observe that SPLR maintains stable performance across a wide range of computation budgets, with adaptive halting consistently matching or outperforming fixed-step inference. Notably, accuracy is largely insensitive to the exact number of refinement steps, suggesting that the model has learned to internalize its reasoning process within the latent refinement dynamics rather than relying on a fixed-length explicit reasoning trace.

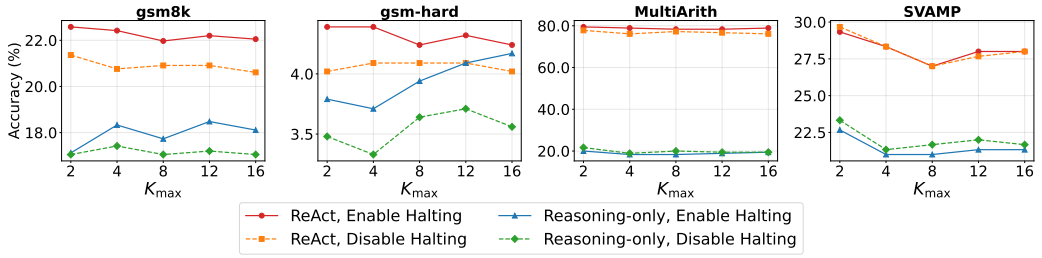

Figure 2: Accuracy versus maximum refinement steps $K_{\max}$ with and without adaptive halting during inference for SPLR.

4.2 Effect of Interaction Patterns: Reasoning-only vs. External Feedback

We next investigate **RQ2**: how different interaction patterns during reasoning—*reasoning-only* versus *ReAct-style with external feedback*—affect reasoning accuracy and generalization.

As shown in Table 1, ReAct-style interaction consistently outperforms reasoning-only supervision across both autoregressive baselines and hierarchical latent reasoning models, with especially pronounced gains on out-of-domain benchmarks such as MultiArith and SVAMP. This suggests that access to external feedback substantially improves robustness beyond what can be achieved by step-level supervision alone.

A key limitation of the reasoning-only setup is the lack of explicit verification for intermediate computations. Because intermediate steps are generated and consumed purely as text, early numerical errors can propagate through subsequent steps, leading to coherent but incorrect final answers. In contrast, ReAct-style interaction anchors each reasoning step to a deterministically verified observation, preventing error accumulation and stabilizing the overall reasoning process.

Reasoning-only vs. ReAct-style

Problem. Paige had 11 songs on her mp3 player. If she deleted 9 old songs and added 8 new songs, how many songs does she have?

Reasoning-only (no external feedback):

- (Step 1) Reasoning: $11 - 9 = 3$ (*incorrect*)
- (Step 2) Reasoning: $3 + 8 = 11$

Final Answer: 11 ×

ReAct-style (with external feedback):

- (Step 1) Action: $11 - 9$ → Observation: 2
- (Step 2) Action: $2 + 8$ → Observation: 10

Final Answer: 10 ✓

Figure 3: Without an external evaluator, reasoning-only models may hallucinate intermediate computation results, leading to error accumulation despite logically consistent subsequent steps.

Figure 3 illustrates this contrast with a representative example. Without external feedback, the model hallucinates an incorrect intermediate result and fails despite logically consistent follow-up reasoning, whereas ReAct-style interaction enforces correct intermediate states and yields the correct final answer.

4.3 Role of Persistent Latent State Propagation

We now address **RQ3**: what role does persistent latent state propagation play in multi-step reasoning during training and inference?

Training-time latent propagation. We first examine latent state propagation during *training*. As described in Section 3.2, we construct an ablated variant that re-initializes the latent state at each reasoning step and instead relies on explicit token-level conditioning over all previous intermediate predictions. As shown in Table 1, this semi-autoregressive variant consistently underperforms models trained with full latent propagation. In particular, for SPLR with ReAct-style interaction, preserving latent propagation yields uniform gains across all benchmarks, with especially large improvements on MultiArith. This indicates that persistent latent states may provide a stronger induc-

Table 2: Inference-time ablation of latent state propagation for SPLR.

Strategy	GSM8K	GSM-Hard	MultiArith	SVAMP
ReAct	22.6	4.4	79.4	29.3
w/o z_h	21.1 *(-1.5)*	4.1 *(-0.3)*	72.8 *(-6.6)*	27.3 *(-2.0)*
w/o z_l, z_h	20.8 *(-1.8)*	4.0 *(-0.4)*	72.8 *(-6.6)*	26.7 *(-2.6)*

tive bias for accumulating reasoning progress across steps, beyond what can be achieved through explicit token-level conditioning alone.

Inference-time latent disruption. We next analyze the effect of disrupting latent state propagation during *inference*. Table 2 reports results where the trained model is evaluated without propagating the high-level state z_h, or without propagating both z_l and z_h. Removing latent propagation leads to consistent accuracy drops across all benchmarks, with the most pronounced degradation on MultiArith, confirming that persistent latent states contribute meaningfully to multi-step reasoning performance.

Interestingly, despite the accuracy drops, the model without latent propagation still solves a substantial number of problems correctly. To investigate this, we perform an instance-level analysis across the three inference-time settings. As shown in Figure 4, the three variants share a large pool of commonly solved problems, yet their solution behaviors diverge considerably. Notably, among problems correctly solved by multiple variants, many are solved via *different reasoning trajectories*: they reach the same final answer but follow distinct sequences of intermediate actions across reasoning steps. Furthermore, the two ablated variants (w/o z_h and w/o z_l, z_h) share **34** commonly solved instances with identical reasoning trajectories that differ from those of the full model, suggesting that removing latent propagation causes the model to converge onto a common default reasoning mode rather than exploring diverse solution strategies. Beyond trajectory divergence, the full-propagation model uniquely solves **72** additional instances that neither ablated variant can solve, while the ablated models produce only **11** and **13** unique solutions respectively, indicating that latent propagation not only diversifies how problems are solved but also expands what can be solved.

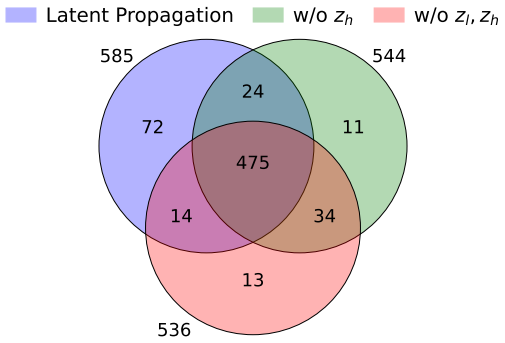

Figure 4: Overlap of correctly solved instances and reasoning trajectory divergence across three inference-time latent propagation settings. Each number denotes the count of instances uniquely or jointly solved.

Overall, these results suggest that latent state propagation primarily affects *how* a problem is solved rather than *whether* it is solved. Rather than being a strict prerequisite for correctness at inference time, latent propagation acts as a *reasoning path stabilizer*, biasing the model toward reusing previously constructed intermediate representations instead of recomputing them from scratch. When latent memory is removed, the model can still reach correct answers for many problems, but does so by discovering alternative solution strategies.

4.4 Effect of Curriculum Learning

We finally study **RQ4**: whether curriculum learning improves the model's ability to learn complex multi-step reasoning. Following Section 3.2, we compare a staged curriculum that gradually increases the maximum number of reasoning steps from 2 to 12 against a setting where this limit is fixed throughout training.

As shown in Table 1 and Figure 5, curriculum learning consistently accelerates convergence and improves final performance across all benchmarks. At the end of training, it yields relative accuracy

gains of approximately **26%** on GSM8K, **22%** on GSM-Hard, **21%** on MultiArith, and **20%** on SVAMP. These results indicate that gradually expanding the reasoning horizon facilitates stable optimization and more effective learning of long-horizon reasoning strategies.

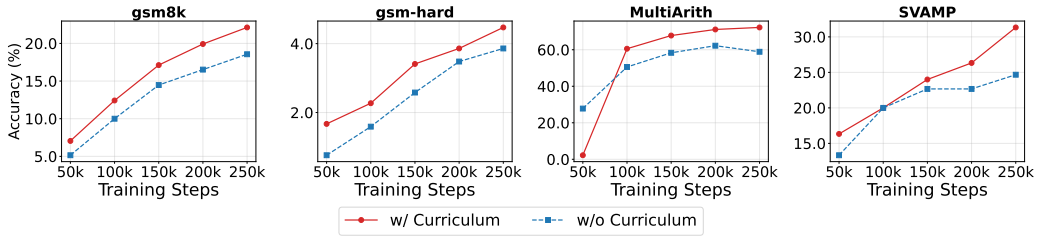

Figure 5: Effect of curriculum learning on training dynamics for **SPLR with ReAct-style interaction**. Progressively increasing the maximum number of reasoning steps leads to faster convergence and higher final accuracy across all benchmarks.

5 Conclusions

We introduced Step-wise Persistent Latent Reasoner (SPLR), a lightweight architecture for explicit multi-step reasoning that propagates a persistent latent state across steps while producing step-wise intermediate hypotheses, optionally in a ReAct-style interaction loop. In a controlled setting where all models are trained from scratch under matched data and optimization budgets, SPLR consistently outperforms standard autoregressive Transformer (GPT-2) baselines across in-domain and out-of-domain math word-problem benchmarks. Our results highlight the benefits of combining hierarchical latent refinement with external feedback, and show that persistent latent state propagation provides a strong inductive bias for accumulating reasoning progress without an ever-growing token history. Looking forward, we see opportunities to extend this framework to broader tool-use and agentic settings, and to combine it with stronger training signals (e.g., distillation or reinforcement learning) to further improve robustness and efficiency.

Acknowledgments

This work was supported in part by Compute Ontario (computeontario.ca) and the Digital Research Alliance of Canada (alliancecan.ca). It was also partially supported by a Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant (RGPIN-2024-04909) and a start up grant from the University of Waterloo.

References

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, et al. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *arXiv preprint arXiv:2507.10524*, 2025.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553380. URL https://doi.org/10.1145/1553374.1553380.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, et al. Do not think that much for 2+ 3=? on the overthinking of o1-like llms. *arXiv preprint arXiv:2412.21187*, 2024.

Zhaoling Chen, Robert Tang, Gangda Deng, Fang Wu, Jialong Wu, Zhiwei Jiang, Viktor Prasanna, Arman Cohan, and Xingyao Wang. Locagent: Graph-guided llm agents for code localization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8697–8727, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Jade Copet, Quentin Carbonneaux, Gal Cohen, Jonas Gehring, Jacob Kahn, Jannik Kossen, Felix Kreuk, Emily McMilin, Michel Meyer, Yuxiang Wei, et al. Cwm: An open-weights llm for research on code generation with world models. *arXiv preprint arXiv:2510.02387*, 2025.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Yuntian Deng, Kiran Prasad, Roland Fernandez, Paul Smolensky, Vishrav Chaudhary, and Stuart Shieber. Implicit chain of thought reasoning via knowledge distillation. *arXiv preprint arXiv:2311.01460*, 2023.

Yuntian Deng, Yejin Choi, and Stuart Shieber. From explicit cot to implicit cot: Learning to internalize cot step by step. *arXiv preprint arXiv:2405.14838*, 2024.

Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. Looped transformers for length generalization. *arXiv preprint arXiv:2409.15647*, 2024.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: a theoretical perspective. *Advances in Neural Information Processing Systems*, 36:70757–70798, 2023.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pp. 10764–10799. PMLR, 2023.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, pp. 11398–11442. PMLR, 2023.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 8154–8173, 2023.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

Alex Havrilla, Yuqing Du, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, Sainbayar Sukhbaatar, and Roberta Raileanu. Teaching large language models to reason with reinforcement learning. *arXiv preprint arXiv:2403.04642*, 2024.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. *arXiv preprint arXiv:2510.04871*, 2025.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*, 2022.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. Mawps: A math word problem repository. In *Proceedings of the 2016 conference of the north american chapter of the association for computational linguistics: human language technologies*, pp. 1152–1157, 2016.

Yann LeCun. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, 62(1):1–62, 2022.

Lucas Lehnert, Sainbayar Sukhbaatar, DiJia Su, Qinqing Zheng, Paul Mcvay, Michael Rabbat, and Yuandong Tian. Beyond a*: Better planning with transformers via search dynamics bootstrapping. *arXiv preprint arXiv:2402.14083*, 2024.

Jindong Li, Yali Fu, Li Fan, Jiahong Liu, Yao Shu, Chengwei Qin, Menglin Yang, Irwin King, and Rex Ying. Implicit reasoning in large language models: A comprehensive survey. *arXiv preprint arXiv:2509.02350*, 2025.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems. *arXiv preprint arXiv:2402.12875*, 1, 2024.

Yubo Ma, Zhibin Gou, Junheng Hao, Ruochen Xu, Shuohang Wang, Liangming Pan, Yujiu Yang, Yixin Cao, Aixin Sun, Hany Awadalla, et al. Sciagent: Tool-augmented language models for scientific reasoning. *arXiv preprint arXiv:2402.11451*, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

Cheng Meixin, Zeqi Zhou, Yijia Xiao, Weihao Xuan, Xiangru Tang, Xu Huang, Tinson Xu, Zerui Xu, Jure Leskovec, Yejin Choi, et al. Vcaf: A multi-agent framework for venture capital decision-making using synthetic startup data. In *NeurIPS 2025 Workshop: Generative AI in Finance*, 2025.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. *arXiv preprint arXiv:2310.07923*, 2023.

Anthony Nguyen and Wenjun Lin. Intra-layer recurrence in transformers for language modeling. *arXiv preprint arXiv:2505.01855*, 2025.

Franz Nowak, Anej Svete, Alexandra Butoi, and Ryan Cotterell. On the representational capacity of neural language models with chain-of-thought reasoning. *arXiv preprint arXiv:2406.14197*, 2024.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*, 2021.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Subhro Roy and Dan Roth. Solving general arithmetic word problems. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pp. 1743–1752, 2015.

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

Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. Codi: Compressing chain-of-thought into continuous space via self-distillation. *arXiv preprint arXiv:2502.21074*, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Wenhui Tan, Jiaze Li, Jianzhong Ju, Zhenbo Luo, Jian Luan, and Ruihua Song. Think silently, think fast: Dynamic latent compression of llm reasoning chains. *arXiv preprint arXiv:2505.16552*, 2025.

Xiangru Tang, Tianrui Qin, Tianhao Peng, Ziyang Zhou, Daniel Shao, Tingting Du, Xinming Wei, Peng Xia, Fang Wu, He Zhu, et al. Agent kb: Leveraging cross-domain experience for agentic problem solving. *arXiv preprint arXiv:2507.06229*, 2025a.

Xiangru Tang, Zhuoyun Yu, Jiapeng Chen, Yan Cui, Daniel Shao, Weixu Wang, Fang Wu, Yuchen Zhuang, Wenqi Shi, Zhi Huang, et al. Cellforge: agentic design of virtual cell models. *arXiv preprint arXiv:2508.02276*, 2025b.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model. *arXiv preprint arXiv:2506.21734*, 2025.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, 2024.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Zehong Wang, Fang Wu, Hongru Wang, Xiangru Tang, Bolian Li, Zhenfei Yin, Yijun Ma, Yiyang Li, Weixiang Sun, Xiusi Chen, et al. Why reasoning fails to plan: A planning-centric analysis of long-horizon decision making in llm agents. *arXiv preprint arXiv:2601.22311*, 2026.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Xilin Wei, Xiaoran Liu, Yuhang Zang, Xiaoyi Dong, Yuhang Cao, Jiaqi Wang, Xipeng Qiu, and Dahua Lin. Sim-cot: Supervised implicit chain-of-thought. *arXiv preprint arXiv:2509.20317*, 2025.

Fang Wu, Xu Huang, Weihao Xuan, Zhiwei Zhang, Yijia Xiao, Guancheng Wan, Xiaomin Li, Bing Hu, Peng Xia, Jure Leskovec, et al. Multiplayer nash preference optimization. *arXiv preprint arXiv:2509.23102*, 2025a.

Fang Wu, Weihao Xuan, Ximing Lu, Mingjie Liu, Yi Dong, Zaid Harchaoui, and Yejin Choi. The invisible leash: Why rlvr may or may not escape its origin. *arXiv preprint arXiv:2507.14843*, 2025b.

Fang Wu, Weihao Xuan, Heli Qi, Ximing Lu, Aaron Tu, Li Erran Li, and Yejin Choi. Deepsearch: Overcome the bottleneck of reinforcement learning with verifiable rewards via monte carlo tree search. *arXiv preprint arXiv:2509.25454*, 2025c.

Peng Xia, Kaide Zeng, Jiaqi Liu, Can Qin, Fang Wu, Yiyang Zhou, Caiming Xiong, and Huaxiu Yao. Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning. *arXiv preprint arXiv:2511.16043*, 2025.

Kevin Xu and Issei Sato. A formal comparison between chain-of-thought and latent thought. *arXiv preprint arXiv:2509.25239*, 2025.

Jian Yang, Wei Zhang, Shawn Guo, Yizhi Li, Lin Jing, Zhengmao Ye, Shark Liu, Yuyang Song, Jiajun Wu, Che Liu, et al. Loopcoder: Scaling code intelligence via looped language models. Technical report, 2026a. URL `https://github.com/IQuestLab/IQuest-Coder-V1/blob/main/papers/LoopCoder_arxiv.pdf`.

Rui Yang, Huitao Li, Weihao Xuan, Heli Qi, Xin Li, Kunyu Yu, Yingjian Chen, Rongrong Wang, Jacques Behmoaras, Tianxi Cai, et al. Toward global large language models in medicine. *arXiv preprint arXiv:2601.02186*, 2026b.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Xinwu Ye, Yicheng Mao, Jia Zhang, Yimeng Liu, Li Hao, Fang Wu, Zhiwei Li, Yuxuan Liao, Zehong Wang, Zhiyuan Liu, et al. Latentchem: From textual cot to latent thinking in chemical reasoning. *arXiv preprint arXiv:2602.07075*, 2026.

Fangxu Yu, Lai Jiang, Haoqiang Kang, Shibo Hao, and Lianhui Qin. Flow of reasoning: Training llms for divergent problem solving with minimal examples. *arXiv preprint arXiv:2406.05673*, 2024.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. Mammoth: Building math generalist models through hybrid instruction tuning. *arXiv preprint arXiv:2309.05653*, 2023.

Zhen Zhang, Xuehai He, Weixiang Yan, Ao Shen, Chenyang Zhao, Shuohang Wang, Yelong Shen, and Xin Eric Wang. Soft thinking: Unlocking the reasoning potential of llms in continuous concept space. *arXiv preprint arXiv:2505.15778*, 2025.

Zhi Zheng and Wee Sun Lee. Beyond imitation: Reinforcement learning for active latent planning. *arXiv preprint arXiv:2601.21598*, 2026.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

A Related Work

Explicit reasoning. Chain-of-thought (CoT) refers to methods that generate an intermediate reasoning process in language before outputting the final answer. This includes prompting (Wei et al., 2022; Khot et al., 2022; Zhou et al., 2022), training models to generate reasoning chains via supervised finetuning (Yue et al., 2023; Yu et al., 2023), and reinforcement learning for reasoning (Wang et al., 2024; Havrilla et al., 2024; Shao et al., 2024; Yu et al., 2024; Wu et al., 2025c). A large body of work improves explicit CoT with decoding and interaction strategies such as self-consistency (Wang et al., 2022), least-to-most prompting (Zhou et al., 2022), reflection-style revision (Shinn et al., 2023; Madaan et al., 2023), and tool-augmented reasoning (Yao et al., 2022; Ma et al., 2024). Recent analyses connect CoT to increased effective depth and expressivity by looping generated outputs back into the Transformer input (Feng et al., 2023; Merrill & Sabharwal, 2023; Li et al., 2024; Nowak et al., 2024). Despite its effectiveness, explicit CoT increases inference cost with longer sequences and often produces redundant or brittle traces, motivating approaches that reduce reliance on long token-level reasoning histories (LeCun, 2022; Hao et al., 2023; Li et al., 2025; Zhang et al., 2025; Xu & Sato, 2025; Wu et al., 2025a).

Thinking in continuous space. Recent work reduces reliance on explicit CoT by moving intermediate computation into continuous representations. Coconut (Hao et al., 2024) switches between language and latent modes, propagating "continuous thoughts", while CODI (Shen et al., 2025) and related approaches use distillation/curriculum to internalize or compress CoT (Deng et al., 2023; 2024; Wei et al., 2025). In parallel, looped Transformer architectures enable iterative hidden-state refinement via recurrence and weight tying (Dehghani et al., 2018; Giannou et al., 2023; Fan et al., 2024; Nguyen & Lin, 2025; Bae et al., 2025), and systems such as LoopLM (Zhu et al., 2025) and LoopCoder (Yang et al., 2026a) connect looping computation with verification/revision behaviors during inference. Complementary to these methods, hierarchical latent refinement models such as HRM (Wang et al., 2025) and TRM (Jolicoeur-Martineau, 2025) iteratively refine predictions with deep supervision. Our SPLR builds on hierarchical refinement but targets explicit multi-step settings: it produces step-wise intermediate hypotheses (optionally with ReAct-style feedback) while propagating a compact latent state across steps, avoiding a growing token history.

B Dataset and Benchmark Details

Step-length distribution and truncation. Both GSM8k-Aug train and validation/test splits provide multi-step solutions as sequences of intermediate equations. Figure 6 summarizes the distribution of the number of reasoning steps (after our ReAct-style `tool_normalized` preprocessing) for train/valid/test. To keep training stable and align with our step-wise architecture, we cap the maximum number of steps to 12 and discard training instances that exceed this limit. After preprocessing, we obtain 357,760 training instances (out of 385,620), 446 validation instances (out of 500), and 1,151 test instances (out of 1,319). We describe the normalization procedure in detail in the following Section B.1.

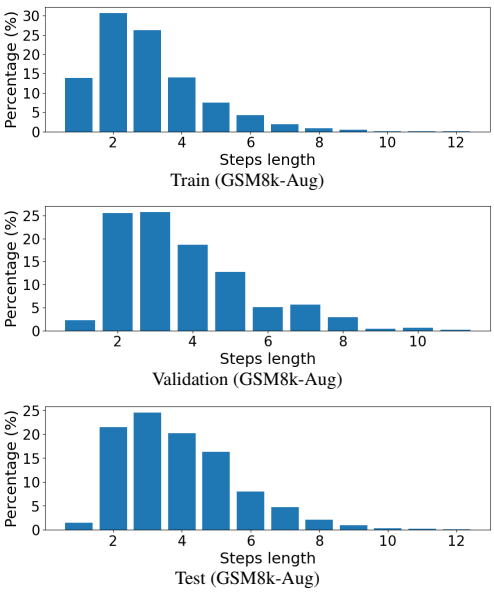

Figure 6: Distribution of reasoning step length (number of intermediate equations) after ReAct-style `tool_normalized` preprocessing.

B.1 Normalized GSM8K-Aug for Reasoning-only and ReAct-style Supervision

Overview. As described in Section 3.1, we train on GSM8k-Aug, which provides a question and a sequence of structured intermedi-

ate math expressions (rather than free-form natural-language rationales). To support our two interaction setups (i.e., *reasoning-only* and *ReAct-style with external feedback*), we apply lightweight normalization to obtain two aligned JSON datasets.

Reasoning-only normalization. For the reasoning-only setting, each sample is represented as {`question, steps, answer`}, where each step contains an equation. We filter and clean samples as follows: (i) we discard empty solutions; (ii) we drop *trivial* steps whose left-hand side contains no arithmetic operator (e.g., `8=8`); and (iii) we require that the final step result matches the provided answer (exact string match after numeric parsing when applicable). If a sample has no remaining valid steps after filtering, it is skipped. For controlled comparisons, we assign a single constant task identifier to all samples (`task_id=0`).

ReAct-style normalization. For the ReAct-style setting, we rewrite each intermediate step into an action–observation pair to simulate tool usage. We first normalize all numbers in questions/steps/answers (e.g., removing commas and canonicalizing decimals such as `4.50`→`4.5` and `.5`→`0.5`). For each equation `lhs=rhs`, we (i) discard trivial equations whose `lhs` contains no operator; (ii) split multi-operator expressions into a sequence of single-operator equations while respecting parentheses and operator precedence; and (iii) validate that each resulting step has a non-empty right-hand side and that the final `rhs` matches the normalized answer (up to numeric tolerance). We skip samples with no valid steps after processing, and (for simplicity) exclude instances with more than 12 resulting steps.

B.2 Benchmark Details

GSM-Hard (Gao et al., 2023) is a harder variant of GSM8K that keeps the same problem statements but replaces numbers with larger/less common values, stressing numerical generalization; it has 1,319 examples matching the GSM8K test size. MultiArith (Roy & Roth, 2015) contains 600 multi-step arithmetic word problems requiring sequencing multiple operations across sentences; each instance is solvable by an equation involving two or more basic operators, and it is commonly used to evaluate compositional arithmetic generalization beyond simpler benchmarks. SVAMP (Patel et al., 2021) is designed to test robustness of math word-problem solvers to superficial variations. It contains 1,000 elementary-level arithmetic problems constructed via controlled transformations (wording/structure/number changes) of existing datasets, and many problems are solvable with short arithmetic expressions (typically no more than two operators).

C Training and Inference Details

C.1 Summary of Training Hyperparameters

Unless otherwise specified, all SPLR models are trained with a global batch size of 512 and totally 400 epochs. We optimize with AdamATan2 ($\beta_1 = 0.9, \beta_2 = 0.95$) using learning rate 5×10^{-5}, 4,000 warmup steps, and weight decay 0.1. We enable the halting loss and set its weight to $q_{\text{halt}} = 0.05$, which is smaller than the default used in prior TRM-style setups (0.5). We initialize SPLR's token embedding matrix from the learned embeddings of pretrained GPT-2. Unless stated otherwise, the backbone configuration uses hidden size 768, 8 attention heads (8 KV heads), intermediate size 2048, and maximum position embeddings 320, with untied input/output embeddings on GPT-2 tokenizer.

For Transformer (GPT-2) baselines, we train all three variants (direct prediction / reasoning-only / ReAct) for 100 epochs and evaluate the checkpoint at 6k training steps. We train with Adam ($\beta_1 = 0.9, \beta_2 = 0.95$), learning rate 5×10^{-4}, cosine learning-rate schedule with 1,000 warmup steps, and weight decay 0.1. Training uses per-device batch size 32 with gradient accumulation 4 on 2 H100 GPUs (bf16).

C.2 Curriculum Learning Schedule

For models trained with curriculum learning, we progressively increase the maximum allowed number of reasoning steps during training. Concretely, we train in four stages with

`max_reasoning_steps` set to 2, 4, 6, and 12, for 100, 100, 100, and 100 epochs, respectively (400 epochs total). All other optimization hyperparameters are kept the same across stages unless otherwise specified.

C.3 Inference-time Configuration and Halting

Different from prior TRM-style setups that often disable learned halting at test time, our model uses the learned halting head at inference and terminates refinement when the predicted q (logits) implies sufficient confidence. During training we set the halting threshold to $\tau = 0$, while at inference we use a stricter threshold $\tau = 0.5$ to enable adaptive computation based on the learned q distribution.

C.4 Task Identifiers and Task Embeddings

This component corresponds to the Puzzle ID / task embedding used in TRM-style setups. In our experiments, we do not enable task identifiers (all train/valid/test instances use `task_id=0`) to focus on learning general-purpose multi-step reasoning rather than exploiting per-task shortcuts. We keep a single task-ID slot in the input interface for compatibility, and use it only to support the halting head that predicts the q logits for adaptive computation.

C.5 Inference Examples

Reasoning-only inference. We illustrate the multi-step reasoning process by SPLR under the reasoning-only setup, where the model does not receive external feedback and generates both expressions and results directly.

Example: Reasoning-only Inference

Input x_0: "Terry eats 2 yogurts a day. They are currently on sale at 4 yogurts for $5. How much does he spend on yogurt over 30 days?"

Reasoning Process:
Step 0 (Initialization):

- Initialize latent state: $s_0 \sim \mathcal{N}(0, I)$
- Initialize: $\hat{y}_0 = \emptyset$

Step 1:

- Update context: $c_1 = \Psi(x_0)$
- Refinement: $s_1 = \mathcal{R}_\theta(c_1, s_0)$
- Decode: $\hat{y}_1 = \texttt{lmhead}(s_1) =$ "2*30=60"

Step 2:

- Update context: $c_2 = \Psi(x_0, \hat{y}_1)$
- Refinement: $s_2 = \mathcal{R}_\theta(c_2, s_1)$
- Decode: $\hat{y}_2 = \texttt{lmhead}(s_2) =$ "60/4=15"

Step 3:

- Update context: $c_3 = \Psi(x_0, \hat{y}_2)$
- Refinement: $s_3 = \mathcal{R}_\theta(c_3, s_2)$
- Decode: $\hat{y}_3 = \texttt{lmhead}(s_3) =$ "15*5=75" with $\langle$eos$\rangle$

Final Answer: Extract from $\hat{y}_3 =$ "75"

ReAct-style inference. We illustrate the multi-step reasoning process by SPLR under the ReAct-style setup using an example from GSM8k-Aug.

Example: ReAct-style Inference

Input x_0: "Toula went to the bakery and bought various types of pastries. She bought 3 dozen donuts which cost \$68 per dozen, 2 dozen mini cupcakes which cost \$80 per dozen, and 6 dozen mini cheesecakes for \$55 per dozen. How much was the total cost?"

Reasoning Process:
Step 0 (Initialization):

- Initialize latent state: $s_0 \sim \mathcal{N}(0, I)$
- Initialize: $\hat{y}_0 = \emptyset$, $o_0 = \emptyset$

Step 1:

- Update context: $c_1 = \Psi(x_0)$
- Refinement: $s_1 = \mathcal{R}_\theta(c_1, s_0)$
- Decode action: $\hat{y}_1 = \texttt{lmhead}(s_1) =$ "3*68"
- Observation: $o_1 =$ "204" (from external evaluator)

Step 2:

- Update context: $c_2 = \Psi(x_0, \hat{y}_1, o_1)$
- Refinement: $s_2 = \mathcal{R}_\theta(c_2, s_1)$
- Decode action: $\hat{y}_2 = \texttt{lmhead}(s_2) =$ "2*80"
- Observation: $o_2 =$ "160"

Step 3:

- Update context: $c_3 = \Psi(x_0, \hat{y}_2, o_2)$
- Refinement: $s_3 = \mathcal{R}_\theta(c_3, s_2)$
- Decode action: $\hat{y}_3 = \texttt{lmhead}(s_3) =$ "6*55"
- Observation: $o_3 =$ "330"

Step 4:

- Update context: $c_4 = \Psi(x_0, \hat{y}_3, o_3)$
- Refinement: $s_4 = \mathcal{R}_\theta(c_4, s_3)$
- Decode action: $\hat{y}_4 = \texttt{lmhead}(s_4) =$ "204+160"
- Observation: $o_4 =$ "364"

Step 5:

- Update context: $c_5 = \Psi(x_0, \hat{y}_4, o_4)$
- Refinement: $s_5 = \mathcal{R}_\theta(c_5, s_4)$
- Decode action: $\hat{y}_5 = \texttt{lmhead}(s_5) =$ "364+330" with $\langle$eos$\rangle$
- Observation: $o_5 =$ "694"

Final Answer: $o_5 =$ "694"

D Model Backbones

Figure 7 visualizes the TRM-style hierarchical refinement backbone used in our main experiments. It maintains low-level (L) and high-level (H) latent components and updates them through alternating refinement and aggregation cycles within each reasoning step. In our implementation, we use `L_cycles=6`, `H_cycles=3`, and `L_layers=2`, where the L-level and H-level Transformer blocks share weights.

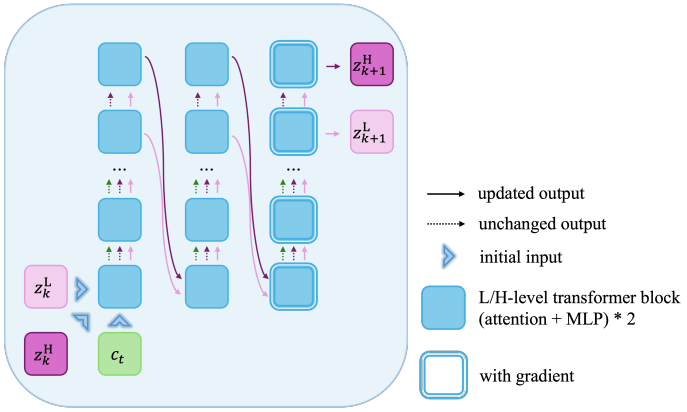

Figure 7: TRM-style backbone used in the main experiments. The backbone maintains low-level (L) and high-level (H) latent components and updates them through alternating refinement and aggregation within each reasoning step. The vertical stack contains 7 Transformer blocks spanning the two components. Here c_t denotes the embedding vectors of the conditioning context c_t.

Table 3: Comparison to post-trained GPT-2 baselines and recent latent/implicit-CoT methods. Results are taken from Zheng & Lee (2026). Best results are highlighted.

	GSM8K	GSM-Hard	MultiArith	SVAMP
CoT-SFT	**52.3**	**12.3**	**94.4**	**58.0**
Answer-SFT	26.4	6.4	49.4	35.3
CoLaR (Tan et al., 2025)	25.7	5.7	86.8	49.9
iCoT (Deng et al., 2024)	30.1	5.7	55.5	29.4
Coconut (Hao et al., 2024)	35.6	8.2	86.1	37.0
SIM-CoT (Wei et al., 2025)	42.2	9.3	87.7	43.9
ATP-Latent (Zheng & Lee, 2026)	42.3	9.8	**94.4**	44.2

E Additional Results

Table 3 reports results from representative post-training approaches that start from a pretrained GPT-2 checkpoint (e.g., CoT-SFT/Answer-SFT and recent implicit/latent-CoT methods). In this comparison, our SPLR models are trained *from scratch* under a controlled budget, and thus are not expected to outperform methods that benefit from large-scale pretraining and subsequent post-training. Nevertheless, SPLR with ReAct achieves 79.4% accuracy on MultiArith, which is reasonably competitive and suggests that step-wise latent refinement can recover substantial arithmetic reasoning ability even without pretraining. Finally, the large gap between CoT-style supervision and direct answer prediction in our baselines indicates headroom for future work that combines our architecture with stronger initialization and post-training signals.