LSRL: Process-Supervised GRPO on Latent Recurrent States Improves Mathematical Reasoning

Anonymous ACL submission

Abstract

Latent-recurrent language models solve tasks by iteratively refining hidden states rather than emitting chain-of-thought tokens, yet the opacity of those hidden trajectories hinders credit assignment and limits mathematical reasoning accuracy. We propose **Latent-State Supervised Reinforcement Learning** (**LSRL**), a process-supervised variant of Guided Reward Policy Optimization that delivers a dense reward at every latent step. Specifically, we decode each of the recurrent depths of a 3.5-billion-parameter Huginn model, score the partial solutions with a GPT-4.1-nano grader aligned to final-answer correctness, and update the policy with LoRA on a single NVIDIA L40S GPU using only **500** GSM-8K training problems. Relative to the depth-8 supervised Huginn baseline, LSRL improves absolute accuracy by **+4.27 points** on GSM-8K and **+2.06 points** on MathQA. These results show that rewarding latent steps is an efficient route to stronger mathematical reasoning in latent-recurrent language models.

1 Introduction

Latent-recurrent language models (LR-LMs) refine an internal state by *looping* a fixed stack of Transformer blocks instead of emitting token-level chains of thought. The recent Huginn model (Geiping et al., 2025) shows that increasing the recurrent depth from $r{=}8$ to $r{=}32$ lifts accuracy on logical-reasoning tasks while keeping the parameter count unchanged—trading parameters for *FLOPs*. Yet the same 3.5-billion-parameter model scores only **13.5**% on the GSM-8K math benchmark at $r{=}8$, far below its logical performance, and the deeper $r{=}32$ variant incurs a four-fold test-time compute cost for a still-modest **24.9**%.

Why does depth help so little? We argue that the bottleneck is *sparse credit assignment*: Huginn's supervised finetuning pipelines apply a single reward to the *final* answer, ignoring the quality of intermediate latent states. By contrast, *process supervision*—rewarding every step of a token-level derivation—has recently improved mathematical reasoning in chain-of-thought models (Lightman et al., 2024; DeepSeek-AI et al., 2025). To date, however, process rewards have **never** been applied to latent states: decoding and grading all r hidden snapshots appears prohibitively expensive.

Our proposal. We introduce *Latent-State Supervised Reinforcement Learning* (**LSRL**), a critic-free GRPO variant that attaches dense rewards to every latent depth of Huginn. A lightweight GPT-4.1-nano grader scores each partial derivation, and LoRA adapters (Hu et al., 2022) update the policy.

Contributions.

- **Algorithmic.** We extend GRPO with *per-depth process rewards*, creating the first process-supervised RL framework for *latent* states.
- **Engineering (I).** We introduce a *one-pass hidden-state cache* that **fully decodes every latent depth**, entire sentences and paragraphs, not token snippets, during a *single* forward/backward pass. This removes the naïve r-fold re-execution and cuts training FLOPs by roughly 50 %.
- **Engineering (II).** We apply **LoRA** adapters to latent-recurrent RL *for the first time*, shrinking trainable parameters by 99 % and enabling single-GPU finetuning.
- **Empirical.** Using only 500 GSM-8K tasks, LSRL lifts Huginn-$r{=}8$ by **+4.27** pp on GSM-8K and **+2.06** pp on MathQA, approaching the $r{=}32$ model while requiring **one-quarter** of its test-time compute.

Road-map. Section 2 reviews latent-recurrent LMs, math-oriented RL, and process supervision.

Section 3 details LSRL; Sections 4–5 present experiments and analysis; Section 6 discusses conclusion and future works.

2 Related Work

Latent recurrence and other compute–accuracy trade-offs. Geiping et al. (2025) introduce *Huginn*, whose recurrent *Core* deepens the network without adding parameters; pushing the depth to r=32 lifts commonsense accuracy but quadruples inference FLOPs. Orthogonal strategies reduce computation in different ways, including sparse Mixture-of-Experts routing (Fedus et al., 2021), RL-learned early-exit policies (Dai et al., 2025), and, most recently, continuous latent policies (Hao et al., 2025). **In contrast**, our method delivers a shaped reward at *every* latent step, eliminating the sparse-gradient bottleneck these approaches leave unresolved.

Reinforcement learning for language models. Outcome-only RLHF commonly relies on PPO (Ouyang et al., 2022; Schulman et al., 2017), while DPO removes the critic through a KL-regularised log-ratio objective (Rafailov et al., 2023). More recent work tackles reward composition and stability (Li et al., 2024) and makes RLHF parameter-efficient via LoRA (Hu et al., 2022), block-wise 8-bit optimisers (Dettmers et al., 2022), QLoRA (Dettmers et al., 2023), PERLHF (Sidahmed et al., 2024), self-rewarding losses (Yuan et al., 2024), and reward distillation (Zhang et al., 2025b). **Instead**, we adopt the critic-free GRPO baseline (DeepSeek-AI et al., 2025) and couple it with latent process rewards, while keeping single-GPU viability through rank-8 LoRA.

Process-supervised reinforcement learning. Dense, token-visible process rewards have proven effective through verifier guidance (Cobbe et al., 2021), automatic step-grading (Lightman et al., 2024), GRPO curricula (DeepSeek-AI et al., 2025), and math-focused systems such as WizardMath (Luo et al., 2023), Improve-Math (Luo et al., 2025), GRPO-LEAD (Zhang and Zuo, 2025), and Efficient-RFT (Shi et al., 2025). Extensions span question decomposition (Chen et al., 2024) and code generation (Ye et al., 2025). Latent-state supervision so far is limited to self-verification probes (Zhang et al., 2025a) or unsupervised latent policies (Hao et al., 2025). **Unlike** these token-visible or probe-based methods, we attach learned process rewards directly to hidden states, avoiding any chain-of-thought decoding.

Mathematical-reasoning resources and curricula. Our evaluation uses GSM-8K (Cobbe et al., 2021), MATH (Lewkowycz et al., 2022), and MathQA (Amini et al., 2019). DeepSeekMath offers a strong open math-centric pre-trained model (Shao et al., 2024), and curriculum-based RL for math has been advanced by WizardMath (Luo et al., 2023), GRPO-LEAD (Zhang and Zuo, 2025), Improve-Math (Luo et al., 2025), and Efficient-RFT (Shi et al., 2025). **Unlike prior work**, Where gains rely on publishing token-level reasoning traces at *inference* time, our approach reaches similar accuracy without any chain-of-thought decoding or storage once the model is trained.

Positioning of our work. *Latent-State RL (LSRL)* uniquely combines (i) dense process rewards, (ii) hidden-state supervision, (iii) recurrent depth, and (iv) parameter-efficient LoRA/QLoRA training, bridging the credit-assignment gap outlined above and delivering improved math reasoning at constant inference cost.

3 Methodology

3.1 Huginn Recap and Notation

Our latent-recurrent language model (LR-LM) follows the PRELUDE–CORE–CODA split of Geiping et al. (2025). During inference the CORE stack R_θ is looped for r iterations while the parameters remain fixed:

$$\mathbf{s}_{k+1} = R_\theta\big([\mathbf{e}; \mathbf{s}_k]\big), \quad k = 0, \ldots, r-1. \tag{1}$$

The final hidden state $\mathbf{s}_r$ is mapped to a token distribution by the shared LM head W_o,

$$p_\theta(y \mid x) = \mathrm{softmax}(W_o\, \mathbf{s}_r). \tag{2}$$

Full-depth decoding. For process supervision, each intermediate latent state $\mathbf{s}_k$ is autoregressively decoded into a textual snapshot $\hat{y}_k$ using the model's Coda and LM head W_o:

$$\hat{y}_k = \mathsf{AutoregressiveDecode}(\mathbf{s}_k), \quad k = 1, \ldots, r, \tag{3}$$

exposing r textual snapshots. These are obtained by processing each cached intermediate state $\mathbf{s}_k$ (see Sec. 3.3.1).

3.2 Latent-State Supervised RL (LSRL)

3.2.1 Reward design

Our reward design evaluates both intermediate reasoning quality and final answer correctness. A lightweight GPT 4.1-nano grader (Section 3.2.2, Appendix B) returns an *internal-quality* score (IQS) and a *math-progress* score (PS) for each decoded snapshot of the latent state $\mathbf{s}_k$. These scores are min–max normalized within each GRPO group to $\hat{q}_k$ and $\hat{p}_k$.

The step-wise reward for snapshot k is

$$R_k \;=\; w_{\text{IQS}}\hat{q}_k + w_{\text{PS}}\hat{p}_k, \tag{4}$$

with $w_{\text{IQS}} = w_{\text{PS}} = 0.5$ as recommended by Yuan et al. (2024).

The process reward aggregates step rewards through a discounted sum:

$$R_{\text{proc}} \;=\; \sum_{k=1}^{r} \gamma^{\,k-1} R_k, \tag{5}$$

where we set $\gamma = 0.99$ following the stable discount used in recent multi-objective RLHF studies (Li et al., 2024).

Finally, we combine the discounted process reward with a binary outcome bonus:

$$R_{\text{tot}} = w_f \,\mathbb{1}[\hat{y}_{\text{final}} = y^*] \;+\; w_p\, R_{\text{proc}}, \tag{6}$$

where $\mathbb{1}[\cdot]$ equals 1 when the predicted answer $\hat{y}_{\text{final}}$ matches the ground truth y^* and 0 otherwise. We fix $(w_f, w_p) = (0.7, 0.3)$, an outcome-dominant split shown effective for mathematical reasoning in Shao et al. (2024) and further supported by the fair-reward study of Li et al. (2024).

3.2.2 Generation workflow and prompt design

To enable process supervision, each relevant intermediate latent state $\mathbf{s}_k$ (for $k = 1, \ldots, r$, where r is the maximum depth for supervision) from a latent trajectory is decoded into a textual snapshot. This decoding is an autoregressive process specific to our LR-LM, leveraging its CODA components and the shared LM head to generate a segment of text reflecting the model's reasoning at depth k.

For every problem, the policy model then generates $G = 8$ complete solution trajectories (final answers). A group of eight offers a good bias–variance trade-off while keeping GPU memory modest, and lies within the 6–8 range adopted in earlier GRPO studies (Shao et al., 2024).

Each of the G trajectories, comprising its r intermediate textual snapshots and its final complete solution text ($\hat{y}_{\text{final}}$), is then evaluated. This evaluation uses GPT-4.1-nano guided by two distinct system prompts (full templates are provided in Appendix B):

1. **Process Grader:** invoked *twice* per trajectory (averaging two independent calls to reduce grader variance). Guided by a unified prompt, the grader assesses each of the r intermediate textual snapshots derived from $\mathbf{s}_1, \ldots, \mathbf{s}_r$. It assigns scores for:
   - *Internal Quality (IQS):* Rates the logical consistency, clarity, and standalone quality of the reasoning in the snapshot.
   - *Mathematical Progress (PS):* Checks if the snapshot meaningfully advances towards solving the problem (e.g., by reducing unknowns, correctly applying an operation, or simplifying the problem state).

   The IQS and PS scores from the two invocations are averaged for each snapshot $\mathbf{s}_k$.

2. **Final Answer Checker:** This grader evaluates the policy model's final generated solution text, $\hat{y}_{\text{final}}$, for correctness against the ground truth y^*, returning $\mathbb{1}[\hat{y}_{\text{final}} = y^*]$.

3.2.3 GRPO objective

Given G sampled trajectories, we first compute the *group-relative* advantage

$$A_i = R_i - \bar{R}, \qquad \bar{R} = \frac{1}{G}\sum_{j=1}^{G} R_j, \tag{7}$$

where $\bar{R}$ is the mini-batch mean reward. We then minimize the clipped loss used by **G**roup-**R**elative **P**olicy **O**ptimisation (GRPO):

$$\mathcal{L}_{\text{GRPO}} = -\sum_{i \in B} \min\Big(\rho_i A_i,\ \text{clip}\big(\rho_i, 1-\varepsilon, 1+\varepsilon\big) A_i\Big) \\ +\ \beta\ D_{\text{KL}}\big(\pi_\theta \,\|\, \pi_{\text{ref}}\big), \tag{8}$$

where $\rho_i = \pi_\theta(y_i \mid x_i) / \pi_{\theta_{\text{old}}}(y_i \mid x_i)$ is the importance ratio.

We adopt $\varepsilon = 0.2$ following the PPO study of Schulman et al. (2017) and the GRPO replication by Shao et al. (2024); this range $(0.1 - 0.3)$ is standard for stable clipped objectives. The KL-penalty coefficient β is controlled by the adaptive

KL scheduler (initial $\beta = 0.1$), adjusted each step to keep $D_{\mathrm{KL}} \approx 0.1$. This setting limits policy drift yet matches the LoRA capacity used in our runs.

Equation (8) is algebraically identical to PPO's surrogate loss; GRPO simply replaces the learned value head with the group mean $\bar{R}$, eliminating the need for a critic network.

3.3 Efficiency Techniques

3.3.1 Efficient Intermediate State Caching for Process Supervision

Our methodology requires evaluating intermediate reasoning steps. The Huginn architecture's recurrent CORE block naturally produces a sequence of latent states $\{\mathbf{s}_k\}_{k=1}^{r}$ (Eq. (1)). These r states are collected and cached in a single forward unrolling of the CORE block. This one-pass generation is highly efficient for obtaining the full set of intermediate states, drastically reducing FLOPs compared to a naive re-execution that would recompute states from $\mathbf{s}_0$ for each depth k. We then perform the decoding process (detailed in Sec. 3.2.2) on each cached state $\mathbf{s}_k$ using the model's CODA components and LM head to generate the textual snapshots required by our PSM graders. The primary FLOP saving highlighted here pertains to the efficient collection of the $\{\mathbf{s}_k\}$ states themselves.

3.3.2 Parameter-Efficient Tuning with LoRA

To fine-tune the Huginn model using our Latent-State Supervised RL (LSRL) approach with manageable computational resources, we employ Low-Rank Adaptation (LoRA) (Hu et al., 2022). Following common practice for effective adaptation (Sidahmed et al., 2024), we inject rank-8 LoRA adapters, with a scaling factor $\alpha = 16$, into specific projection matrices within Huginn's recurrent CORE block. Specifically, adapters are applied to the query/key/value and output projections of the attention mechanism, as well as the up-projection and down-projection layers of the MLP. This strategy resulting in only 0.17% trainable parameters.

3.4 Training Loop Overview

Algorithm below condenses one GRPO update in our *Latent-State RL* (LSRL) pipeline. For the interested reader, Appendix A presents detailed pseudocode implementation and Appendix B provides the GPT-4.1-nano grading configuration.

1. **Sample roll-outs.** For each prompt in the mini-batch, nucleus-sample G trajectories.

2. **Generate intermediate textual snapshots.** For each of the G trajectories, from its efficiently cached intermediate latent states $\{\mathbf{s}_k\}$ (Sec. 3.3.1), generate a textual snapshot for each relevant depth k via autoregressive decoding (detailed in Sec. 3.2.2).

3. **Grade snapshots.** Send each depth-k string to the GPT-4.1-nano graders (Sec. 3.2.2) and collect scores $\{\mathrm{IQS}_k^{(g)}, \mathrm{PS}_k^{(g)}\}$.

4. **Compute rewards and advantages.** Fuse snapshot and final scores into a total reward $R_{\mathrm{tot}}^{(g)}$ for each trajectory (g) via Eqs. (4)–(6), then form group-relative advantages with Eq. (7).

5. **Optimize.** Minimize the GRPO loss (Eq. (8)) using the AdamW optimiser.

6. **Update parameters.** Apply rank-8 LoRA deltas to the 16 CORE projection matrices only (Sec. 3.3.2); all other base model weights stay frozen.

4 Experiment Design and Setup

4.1 Tasks and Datasets

We assess mathematical reasoning on three open benchmarks:

- **GSM-8K** grade-school word problems (Cobbe et al., 2021),
- **MATH** ("Minerva-MATH") theorem-style proofs (Lewkowycz et al., 2022),
- **MathQA** multi-step arithmetic questions (Amini et al., 2019).

For each dataset we use its official test split for evaluation and do not include any test items in training. The reinforcement-learning phase fine-tunes on a random subset of 500 problems from the GSM-8K training split; no synthetic data or curriculum-generated examples are added.

4.2 Model Variants

We evaluate four systems, summarised in Table 1:

- **Huginn-SFT-r8**: supervised baseline, recurrent depth r=8.
- **RL-Outcome**: depth-8 model fine-tuned with GRPO using a single final-answer reward.

Name	Params	r	Trainable %[1]	Reward
SFT-r8	3.5 B	8	0	–
RL-Outcome	3.5 B	8	0.17	Final
LSRL	3.5 B	8	0.17	Final+PSM
SFT-r32	3.5 B	32	0	–

Table 1: Variants compared in this study.

Parameter	Value
Optimizer	AdamW
Learning rate	2×10^{-6} (constant)
Trajectories G	8
Discount γ	0.99
Clip ε	0.2
KL target β	0.1 (adaptive)
LoRA rank / α	8 / 16
Quantisation	int8 (QLoRA)

Table 2: Core hyper-parameters.

- **LSRL (ours)**: depth-8 model trained with both final and process (PSM) rewards.

- **Huginn-SFT-r32**: deeper supervised baseline, $r{=}32$.

All variants start from the public *Huginn-3.5B* checkpoint and are updated with rank-8 LoRA adapters; thus fewer than 0.2 % of parameters are trainable in the RL runs.

4.3 Training and Evaluation Procedure

We optimize with AdamW at a constant learning rate of 2×10^{-6}. To process 32 unique prompts before each weight update, gradients are accumulated over four sequential micro-batches; each micro-batch handles trajectories generated from 8 unique prompts. For every unique prompt, $G{=}8$ distinct trajectories are produced by the policy model. In our LSRL model, which incorporates process rewards, the intermediate reasoning steps within these trajectories are evaluated by a Problem-Specific Model (PSM) based on GPT 4.1-nano, and the resulting step-wise rewards are discounted by $\gamma_{psm}{=}0.99$.

At test time we decode greedily ($T{=}0$) and report the **pass@1** metric across GSM-8K, MATH and MathQA benchmarks. All runs fit on a single NVIDIA L40S-class GPU in `int8` mode using QLoRA (Dettmers et al., 2023).

5 Results and Discussion

5.1 Accuracy and Efficiency

Overall gains. Relative to the supervised depth-8 baseline (*SFT-r8*), our latent-state supervised RL model (*LSRL*) raises accuracy by **+4.27 points** on GSM-8K, **+1.33 points** on MATH, and **+2.06 points** on MathQA. These improvements indicate that process-level rewards substantially strengthen Huginn's mathematical reasoning.

The improvement on MATH is modest compared with grade-school datasets for several reasons. First, the problems are intrinsically harder: proofs often span hundreds of tokens, require symbolic manipulation, or invoke high-level tactics such as case splits and geometric constructions that are absent from GSM-8K and MathQA. Second, the RL phase fine-tunes on just 500 GSM-8K items; the heuristics learned there, mainly short arithmetic chains, transfer only partially to Olympiad notation and LaTeX-formatted derivations. Third, our reward model scores local algebraic progress, so higher-order reasoning steps that are essential for MATH remain largely invisible to the shaping signal.

One natural remedy is difficulty-aware curriculum learning. Adaptive schedulers such as AdaRFT (Shi et al., 2025) sample problems whose estimated difficulty sits just beyond the model's current competence, accelerating PPO-style fine-tuning on mathematical reasoning. GRPO-LEAD (Zhang and Zuo, 2025) shows that re-weighting the advantage term by problem difficulty further sharpens GRPO updates. Interleaving easier GSM items with progressively harder MATH subsets, or replacing the generic grader with a proof-validity scorer, should expose Huginn to richer reasoning traces while preserving the dense feedback that proved effective on grade-school tasks.

Source of the improvement. Outcome–only RL lifts GSM-8K by only **+1.05 pp**. The **+3.22 pp** additional gain realized after adding process supervision therefore contributes roughly **75 %** of the total lift. Two quantitative diagnostics corroborate that dense stepwise rewards, not merely extra policy-gradient updates, drive this gap.

That said, the outcome-only baseline is not completely ineffective. Following the "posterior sharpening" explanation in DeepSeek-R1 (DeepSeek-AI

[1]Percentage of total model parameters updated during fine-tuning (LoRA adapters only).

Model	r	GSM (%)	MATH (%)	MathQA (%)	FLOPs/tok[†]
SFT-r8	8	13.49	5.61	24.07	1.0×
RL-Outcome	8	14.54	6.32	24.62	1.0×
LSRL (ours)	8	**17.76**	**6.94**	**26.13**	1.0×
SFT-r32	32	24.87	11.24	27.97	4.0×

Table 3: **Accuracy and compute.**[†]FLOPs per token grow linearly with recurrent depth; depth–8 is normalised to 1.0×.

et al., 2025), even a binary reward moves probability mass away from trajectories that end with off-by-one arithmetic slips, yielding the modest +1 pp boost we observe. Nonetheless, because the reward is observed only after all eight latent iterations, credit assignment remains long-horizon, and improvement quickly saturates.

Compute–depth trade-off. Having established where the accuracy gain comes from, we next investigate how much compute it saves. Although depth-32 inference still achieves the highest raw accuracy, *LSRL-r8* recovers approximately **75 %** of the GSM-8K score while consuming only **25 %** of the recurrent compute (1.0 × vs. 4.0 × FLOPs).

Looking forward, repeating the LSRL recipe at larger depths (r=16 or 32) appears especially promising. A deeper Huginn exposes up to four times as many latent snapshots, and each additional snapshot supplies an independent reward signal. Moreover, the depth-32 supervised baseline already achieves the best raw accuracy, suggesting that that process-supervised RL at $r \geq 16$ could close much of the remaining MATH gap while retaining strong accuracy-per-FLOP profile demonstrated at r=8.

5.2 Qualitative Trajectory Analysis

Figure 1 decodes the latent states s_k $(k=1...8)$ for a representative GSM-8K problem under the baseline *SFT-r8* and our *LSRL*. The baseline drifts off-topic as early as depth 1, showing hallucinating boilerplate phrases and incoherent arithmetic, while *LSRL* produces a correct plank count at depth 1, refines it consistently, and arrives at the gold answer ($12 000) by depth 8. This contrast illustrates how process supervision rewards guide the policy toward higher-quality trajectories long before the final step.

Trajectory selection. Because the reward is observed at *every* depth, the policy quickly abandons low-reward paths, such as those producing nonsensical tokens, so probability mass concentrates on trajectories whose early states look algebraically plausible.

Human-legible reasoning. The reward model is a *GPT-4.1-nano* grader trained on human-style Chain-of-Thought data; latent states therefore evolve toward fluent, step-by-step explanations that are easy for humans (and automatic verifiers) to follow.

Noise suppression. Process supervision sharply reduces "garbled-symbol" segments. A related observation was made by Luo et al. (2025), who show that automated process supervision eliminates meaningless or redundant tokens while boosting math accuracy; our results mirror this effect inside Huginn's latent loop.

Residual issues analysis. While *LSRL* removes most garbled tokens, three problems persist:

1. **Prompt-echo noise** at depth 1 ("*first 300 seconds is not enough ...*"), showing the model still copies stray prompt fragments before it settles on algebra.
2. **Qualitative fluff** at depth 2 ("*astronomically large number*"), a logically irrelevant phrase.
3. **Unit/type mix-ups** at depth 3: revenue is computed on planks rather than tables, yielding an off-by-factor answer ($222 000).

These errors persist for two potential reasons. First a *granularity mismatch*: the reward scores the entire paragraph at each depth, so brief fluff can be averaged out. Second, a *capacity limit*: the lightweight GPT-4.1-nano judge occasionally overestimates unit-mismatched arithmetic, allowing the policy to "game" the reward early in the trajectory.

Further improvement paths. **(a) Fine-tune the existing GPT-4.1-nano reward model.** Iteratively re-labeling fresh trajectories and distilling both *responses* and *rewards* has been shown to let ∼1 B-parameter judges outperform larger teachers (Zhang et al., 2025b). Applying the same "reward-distillation" loop every few RL epochs should tighten early depth supervision at negligible compute cost.

Q: *John chops down 30 trees. From each tree, he can make 25 planks. A table takes 15 planks to make and sells for $300. He paid $3000 for all the labor. How much profit did he make?*

Answer: *$12,000*

Baseline SFT-r8

Depth-1: First, let's start with some basic concepts and some basic concepts. *****Please follow the step-click**** ****Time-time-time*******

Depth-2: John chops **15 trees** to make $3000 a total of **$300,000**. He pays $3000 for each tree, $300 for a total of $3000. Each tree is $300. How much money does he make?

Depth-3: John made **30 planks from each tree** and 25 planks from the remaining 25 trees. So, he made $30 \times 30 = 150$ **planks** in total. John made $25 \times 15 = 30$ **planks** from the first 25 trees and $25 \times 15 = 45$ **planks** from the remaining 25 trees. John's total profit is $300 \times 30 = 900 \times 15 = \$3000000$.

... intermediate depths omitted ...

Depth-8 (final): To solve this problem, we need to calculate the total number of planks John has and then determine the profit he made from the sales of the planks.

Step 1: Calculate the total number of planks John has. John has 30 trees and each tree produces 25 planks. So, the total number of planks John has is: 30 trees * 25 planks / tree = 750 planks
Step 2: Calculate the total number of tables John sold. John sold tables for $300 per table, and **he sold 750 tables**. So, the total number of tables John sold is: **750 tables**
Step 3: Calculate the profit John made from the sales of the planks. John paid $3000 for all the labor and each table takes 15 planks to make. So, the total number of planks John used for the tables is: **750 tables * 15 planks/table = 11250 planks**
Step 4: The profit John made from the sales of the planks is: Profit = (Profit per table * Number of tables) * Number of planks used for tables = **($300 * 750) * 11250 = $2**.

LSRL (PSM RL)

Depth-1: John chops down 30 trees. From those trees he makes **750 planks**. A table takes 15 planks. Each plan cuts the tree down to planks, then repeat iteratively. The answer: **first 300 seconds is not enough to compute the total**.

Depth-2: To calculate the profit, John needs to calculate the labor cost ($3,000) and the selling price of each table. The total cost of the trees is **an astronomically large number**.

Depth-3: He can make 25 planks. A table takes 15 planks to make and sells for $300. He paid $3000 for all the labor. How much profit did he make? Huginn

John can make 25 planks from each tree, so he can make $30 \times 25 = 750$ **planks**. He sold each **plank** for $300, so revenue is $750 \times 300 = \$225{,}000$. He paid $3000 for all the labor. John's profit is **$225,000 – $3,000 = $222,000**.

... intermediate depths omitted ...

Depth-8 (final): He can make 25 planks. A table takes 15 planks to make and sells for $300. He paid $3000 for all the labor. How much profit did he make? Huginn

1. Calculate the total number of planks John made: from the 30 trees, he made $30 \times 25 = 750$ **planks**.
2. Calculate the total number of tables sold: $750 \div 15 = 50$ **tables**.
3. Calculate the total revenue from the planks: Each table sold for 300, so the total revenue from tables is $50 \times \$300 = \$15{,}000$.
4. Calculate the profit: To find the profit, subtract the cost from the revenue: **$15,000 - $3,000 = $12,000**.

Legend: **Hallucinations/Off-topic** **Calculation Errors** **Correct Calculations** **Residual Issues**

Figure 1: Decoded latent-state snapshots for baseline SFT-r8 vs. LSRL on a GSM-8K example. Early depths from *LSRL* already contain correct computations, whereas the baseline exhibits off-topic drift and calculation errors.

(b) Train a specialised process reward model (PRM). Chen et al. (2024) demonstrate that an auto-generated, symbol-aware PRM which verifies each sub-question yields larger math gains than text-only graders. Plugging such a PRM in place of GPT-nano would directly penalize the unit/type errors still visible at depths 2–3 and can be combined with via a two-stage curriculum: first clean syntax with nano, then enforce symbolic correctness with PRM.

5.3 Shallow-Recurrence Ablation

We reran the entire training recipe with the latent loop shortened to four iterations and evaluated on GSM-8K and MathQA. Table 4 reveals a **flat plateau**: every variant clusters around 8% on GSM-8K and 23% on MathQA. This stark contrast with our $r{=}8$ findings suggests a minimum threshold of recurrent depth is necessary before process supervision can take effect. At $r{=}4$, the

Model	GSM	MathQA
SFT-r4	8.36	22.93
RL-Outcome-r4	8.01	22.47
LSRL-r4	**8.59**	**23.14**

Table 4: **Shallow-loop ablation (depth $r{=}4$).** MATH is omitted because all variants score $\leq 1\%$, rendering the task trivial at this depth.

intermediate latent states likely contain insufficient meaningful reasoning steps to provide an effective grading signal, resembling attempts to grade underdeveloped work. These results highlight that recurrence depth plays a critical role not only in raw model performance but also in enabling effective reinforcement learning. The stark contrast between ineffective reinforcement at $r{=}4$ and substantial gains at $r{=}8$ suggests a threshold of required computational depth for LSRL, though further work is needed to precisely characterize this boundary.

6 Conclusion and Future Work

We introduced Latent-State Supervised Reinforcement Learning (LSRL), a process-supervised variant of GRPO that delivers dense rewards at each latent iteration of recurrent language models. Our approach addresses a key limitation of latent-recurrent models: while they can achieve impressive reasoning capabilities with fewer parameters, their opaque hidden trajectories hinder effective credit assignment, especially for mathematical reasoning tasks.

Our work makes several contributions to latent reasoning in language models. First, we developed a novel framework for process-supervised RL that operates on latent states rather than explicit tokens, creating the first process-supervised approach for latent recurrence. Second, we introduced technical innovations that make this approach practical: a one-pass hidden-state cache that fully decodes every latent depth in a single forward/backward pass, and LoRA adapters for efficient fine-tuning of latent-recurrent RL. Finally, using only 500 GSM-8K training problems and a single GPU, we demonstrated substantial improvements of **+4.27** points on GSM-8K and **+2.06** points on MathQA, approaching the performance of models with 4× the compute requirements.

Our results demonstrate that latent-recurrent architectures offer a promising alternative path to scaling reasoning capabilities in language models. While most approaches focus on either increasing parameter count or extending inference through chain-of-thought tokens, LSRL enables models to scale through test-time computation in the latent space. This provides several advantages: (1) reduced memory requirements during training and inference, and (2) no need for specialized training data containing intermediate reasoning steps.

Future directions. There are several promising directions for extending this work:

1. **Scaling to larger recurrent depths:** Applying LSRL to deeper models ($r{=}16$ or $r{=}32$) should yield additional gains, as these models expose more latent states for supervision while maintaining the parameter efficiency advantage.
2. **Specialized reward models:** Developing mathematical process reward models that are symbol-aware and can verify intermediate algebraic steps would address the unit/type confusion issues we observed and potentially close more of the gap on the MATH benchmark.
3. **Curriculum learning:** Implementing difficulty-aware curricula like AdaRFT or GRPO-LEAD could accelerate learning and improve transfer to more complex domains by progressively exposing the model to harder reasoning problems.
4. **Cross-domain transfer:** Extending LSRL beyond mathematics to domains such as logical reasoning, coding, and causal inference could reveal whether the process-supervision benefits generalize across different reasoning types.

We believe that LSRL represents an important step toward more efficient mathematical reasoning in language models. By aligning the optimization process with the recursive structure of latent-recurrent models, we achieve performance that would typically require significantly more parameters or deeper recurrence depth. This suggests that process-level supervision in the latent space is a promising direction for developing more capable yet efficient reasoning systems.

Limitations

While LSRL demonstrates promising results in improving mathematical reasoning in latent-recurrent language models, our approach has several limitations that should be addressed in future work.

Training data limitations. Our reinforcement learning phase relies on only 500 GSM-8K training problems, which represents a small fraction of the available mathematical content. This limited training set may restrict the diversity of problem-solving strategies that the model learns. Additionally, we did not implement curriculum learning or generate synthetic training data, which could potentially improve performance on harder problems. Our approach also lacks exposure to complex mathematical domains like multi-step proofs, geometric reasoning, or higher-level algebra, which may explain the more modest improvements on the MATH benchmark compared to grade-school datasets.

Reward model limitations. The GPT-4.1-nano grader we use as a reward model has inherent limitations in its mathematical understanding. Unlike specialized symbolic verifiers, it may struggle to detect subtle errors in calculations or logical steps, particularly for complex mathematical operations. This could lead to reward misalignment where the model is reinforced for mathematically incorrect but plausible-sounding reasoning.

There is also potential for reward hacking in our approach. The model might learn to optimize for superficial features that correlate with higher rewards without truly improving its reasoning capabilities. For example, it might learn to use particular phrasing or formatting that the grader tends to score highly, rather than developing deeper mathematical understanding.

Quality and consistency issues. As discussed in Section 5.2, several quality issues persist in LSRL's outputs:

1. **Prompt-echo phenomena:** The model still exhibits a tendency to copy portions of the input prompt at early depths, suggesting incomplete decoupling of input processing and reasoning initialization.
2. **Superfluous content:** The presence of qualitative "fluff" and logically irrelevant phrases at intermediate depths indicates that the model has not fully learned to focus on mathematically relevant reasoning steps.
3. **Semantic confusion:** Unit/type mix-ups in numerical reasoning (e.g., conflating planks with tables or misapplying operations to incorrect entities) shows remaining weaknesses in the model's conceptual understanding of applied mathematics.

These issues suggest that while process supervision improves overall performance, it does not completely solve the underlying challenges in mathematical reasoning.

Generalization limitation. Our approach shows stronger improvements on grade-school arithmetic (GSM-8K, MathQA) than on competition-level mathematics (MATH). This indicates challenges in generalizing from simpler computational patterns to more advanced mathematical reasoning. The transfer learning path from GSM-8K's word problems to MATH's formal notation appears to be limited, suggesting that separate training on symbolic mathematics might be necessary.

Additionally, we have only evaluated LSRL on a narrow set of mathematical reasoning tasks. Its effectiveness for other reasoning domains, such as programming, causal reasoning, or planning problems, remains unexplored. This scope limitation makes it difficult to assess whether the improvements we observe are specific to arithmetic reasoning or represent a more general enhancement to latent-recurrent models' reasoning capabilities across domains.

Ethical Considerations

Intended use and reliability. LSRL is released for research on improving mathematical reasoning capability. While LSRL shows improved performance on mathematical benchmarks, latent-recurrent language models still make errors even on elementary calculations. Applications of these models to domains requiring mathematical precision should include appropriate verification mechanisms, especially for safety-critical applications. Our work does not claim to eliminate the need for human oversight in mathematical reasoning tasks, and users of such systems should be aware of these limitations to prevent over-reliance on probabilistic models for deterministic mathematical problems.

Resource efficiency. Our approach demonstrates that smaller models (3.5B parameters) with appropriate supervision can approach the performance of larger or computationally more expensive models on mathematical reasoning tasks. This has positive implications for both environmental impact and accessibility. By focusing on efficient test-time compute scaling rather than parameter scaling alone, LSRL potentially reduces the energy consumption associated with training large mathematical reasoning models. Additionally, the ability to deploy smaller yet capable models could democratize access to mathematical reasoning capabilities across a wider range of hardware constraints.

Data privacy and bias. Our training approach relies on GSM-8K, and all benchmarks used for evaluation are publicly available datasets that contain no personal or sensitive information. However, we acknowledge that these datasets primarily reflect grade school level mathematics curricula and are exclusively in English. This limitation could affect model performance on problems formulated at different difficulty level or in different languages. We plan to address this limitation by curating more diverse and multilingual mathematical problem sets in future research.

References

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. MathQA: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2357–2367, Minneapolis, Minnesota. Association for Computational Linguistics.

Zhaorun Chen, Zhuokai Zhao, Zhihong Zhu, Ruiqi Zhang, Xiang Li, Bhiksha Raj, and Huaxiu Yao. 2024. AutoPRM: Automating procedural supervision for multi-step reasoning via controllable question decomposition. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1346–1362, Mexico City, Mexico. Association for Computational Linguistics.

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. 2021. Training verifiers to solve math word problems. arXiv:2110.14168. https://doi.org/10.48550/arXiv.2110.14168.

Muzhi Dai, Chenxu Yang, and Qingyi Si. 2025. S-GRPO: Early exit via reinforcement learning in reasoning models. arXiv:2505.07686.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, and et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. arXiv:2501.12948. https://doi.org/10.48550/arXiv.2501.12948.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 2022. 8-bit optimizers via block-wise quantization. In *International Conference on Learning Representations*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. In *Thirty-seventh Conference on Neural Information Processing Systems*.

William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. In *Journal of Machine Learning Research*.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. 2025. Scaling up test-time compute with latent reasoning: A recurrent depth approach. arXiv:2502.05171. https://doi.org/10.48550/arXiv.2502.05171.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2025. Training large language models to reason in a continuous latent space. arXiv:2503.01234. https://openreview.net/forum?id=tG4SgayTtk.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*.

Aitor Lewkowycz, Anders Johan Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Venkatesh Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, and 1 others. 2022. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems*.

Jiahui Li, Hanlin Zhang, Fengda Zhang, Tai-Wei Chang, Kun Kuang, Long Chen, and Jun Zhou. 2024. Optimizing language models with fair and stable reward composition in reinforcement learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 10122–10140. Association for Computational Linguistics.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations (ICLR)*.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, and 1 others. 2023. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. arXiv:2308.09583. https://doi.org/10.48550/arXiv.2308.09583.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Lei Meng, Jiao Sun, and Abhinav Rastogi. 2025. Improve mathematical reasoning in language models with automated process supervision. arXiv:2406.06592.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Gray, and 1 others. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*.

Robert Rafailov, Andrew Wilson, and Stefano Ermon. 2023. Direct preference optimization: Your language model is secretly a reward model. In *International Conference on Learning Representations*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. arXiv:1707.06347.

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

Taiwei Shi, Yiyang Wu, Linxin Song, Tianyi Zhou, and Jieyu Zhao. 2025. Efficient reinforcement finetuning via adaptive curriculum learning. *arXiv preprint arXiv:2504.05520*.

Hakim Sidahmed, Samrat Phatale, Alex Hutcheson, Zhuonan Lin, Zhang Chen, Zac Yu, Jarvis Jin, Simral Chaudhary, Roman Komarytsia, Christiane Ahlheim, Yonghao Zhu, Bowen Li, Saravanan Ganesh, Bill Byrne, Jessica Hoffmann, Hassan Mansoor, Wei Li, Abhinav Rastogi, and Lucas Dixon. 2024. Parameter-efficient reinforcement learning from human feedback. arXiv:2403.10704.

Yufan Ye, Ting Zhang, Wenbin Jiang, and Hua Huang. 2025. Process-supervised reinforcement learning for code generation. arXiv:2502.01715.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason E. Weston. 2024. Self-rewarding language models. In *Forty-first International Conference on Machine Learning*.

Anqi Zhang, Yulin Chen, Jane Pan, Chen Zhao, Aurojit Panda, Jinyang Li, and He He. 2025a. Reasoning models know when they're right: Probing hidden states for self-verification. arXiv:2504.05419.

Jixiao Zhang and Chunsheng Zuo. 2025. Grpo-lead: A difficulty-aware reinforcement learning approach for concise mathematical reasoning in language models. *arXiv preprint arXiv:2504.09696*.

Yudi Zhang, Lu Wang, Meng Fang, Yali Du, Chenghua Huang, Jun Wang, Qingwei Lin, Mykola Pechenizkiy, Dongmei Zhang, Saravan Rajmohan, and Qi Zhang. 2025b. Distill not only data but also rewards: Can smaller language models surpass larger ones? arXiv:2502.19557. https://doi.org/10.48550/arXiv.2502.19557.

A Pseudocode: PSM-Guided GRPO Training of Huginn

Implementation notes.

Algorithm 1: GRPO with process-supervised (PSM) rewards at every latent step. The one-pass hidden-state cache avoids re-executing the recurrent core during DecodeSnapshot.

Input: Mini-batch of prompts $\{\mathbf{x}^{(i)}\}_{i=1}^{B}$ and gold answers $\{y^{(i)}\}_{i=1}^{B}$

Output: Updated Huginn parameters $\theta \leftarrow \theta'$

Hyper-params: recurrent depth $r{=}8$; # trajectories per prompt G; reward weights $\lambda_{\text{final}}, \lambda_{\text{IQS}}, \lambda_{\text{PS}}$; discount γ.

1. Policy rollout

for $i \leftarrow 1$ **to** B **do** `// vectorised across` $B \times G$ `trajectories`

  for $g \leftarrow 1$ **to** G **do**

    $\{\mathbf{s}_{1:r}^{(i,g)}\} \leftarrow \text{HuginnForward}(\mathbf{x}^{(i)}, \theta)$

    $\hat{y}^{(i,g)} \leftarrow \text{DecodeFinal}(\mathbf{s}_r^{(i,g)})$

    for $k \leftarrow 1$ **to** r **do**

      $\hat{t}_k^{(i,g)} \leftarrow \text{DecodeSnapshot}(\mathbf{s}_k^{(i,g)})$ `// cached for reward`

    end

  end

end

2. External grading (batched API calls to GPT-4.1 nano)

foreach *trajectory* (i, g) **do**

  $\{q_k^{\text{IQS}}, q_k^{\text{PS}}\}_{k=1}^{r} \leftarrow \text{GradeSeq}(\{\hat{t}_k^{(i,g)}\}_{k=1}^{r})$

  $q^{\text{final}} \leftarrow \text{GradeAnswer}(\hat{y}^{(i,g)}, y^{(i)})$

  $R^{(i,g)} \leftarrow \lambda_{\text{final}}\, q^{\text{final}} + \lambda_{\text{IQS}} \sum_{k=1}^{r} \gamma^{k-1} q_k^{\text{IQS}} + \lambda_{\text{PS}} \sum_{k=1}^{r} \gamma^{k-1} q_k^{\text{PS}}$

end

3. GRPO update (critic-free PPO)

$\nabla_\theta J \approx \frac{1}{BG} \sum_{i,g} \big[\, \pi_\theta(\hat{y}^{(i,g)} \,|\, \mathbf{x}^{(i)}) - \pi_{\theta_{\text{old}}}(\hat{y}^{(i,g)} \,|\, \mathbf{x}^{(i)}) \big]\, \text{clip}\big(R^{(i,g)} - \bar{R},\, -\epsilon,\, +\epsilon\big)$ `// standard GRPO ratio clip`

$\theta \leftarrow \theta - \eta\, \nabla_\theta J$ `// AdamW`

- *Hidden-state cache:* we store the sequence $\mathbf{s}_{1:r}$ during the forward pass and reuse it for all snapshot decodings, avoiding r additional core executions.
- *LoRA updates:* only the recurrent CORE projection matrices receive rank-8 LoRA adapters; Prelude/Coda blocks remain frozen.
- *Parallel grading:* calls to GPT-4.1 nano for IQS/PS and final correctness are issued asynchronously to maximise throughput.

B GPT-4.1 Nano PSM-Grader Configuration

To generate dense, machine-parseable rewards we invoke two `gpt-4.1-nano` instances:

- **Graders A/B** – called twice per trajectory and

Parameter	Graders A/B	Grader C
Model ID	`gpt-4.1-nano`	
Temperature	0.0	0.0
Top-p	1.0	1.0
Max tokens	3000	200
Response format	{*type*: "json_object"}	

Table 5: API settings used for all PSM grading calls.

averaged, they assess *Intrinsic Quality* and *Progress* of every intermediate snapshot.

- **Grader C** – checks the numerical correctness of the *final* answer.

B.1 Generation hyper-parameters

B.2 System prompt — Process-quality Graders (A/B)

```
You are a meticulous and objective AI assistant specializing
     in evaluating mathematical reasoning. You will be given
     a math problem, its ground truth numerical final answer,
      and a series of up to `max_depth_generated` solutions (
     Output_1 to Output_`max_depth_generated`) generated by a
      student AI at increasing "reasoning depths".

Your task is to evaluate EACH of the outputs individually and
     relative to the previous one. For each Output_k (where k
      is from 1 to `max_depth_generated`), assign scores on a
      0.0 to 2.0 scale as described below:

1. **Extract Final Numerical Answer (EFA_k):** From Output_k,
     extract the final numerical answer. If no clear
     numerical answer is present, state "None".
2. **Assess Correctness (AC_k):** Compare EFA_k with the
     Ground Truth Final Answer. State if it's "Correct" or "
     Incorrect".

3. **Intrinsic Quality Score (IQS_k):** Assign a score from
     0.0 to 2.0 based on the standalone quality of Output_k.
   * **0.0 to 0.3:** Severely flawed.
   * **0.4 to 0.7:** Significantly flawed.
   * **0.8 to 1.2:** Mixed quality / Neutral.
   * **1.3 to 1.6:** Fair quality.
   * **1.7 to 1.9:** Good quality.
   * **2.0:** Excellent quality.

4. **Progress Score (PS_k):** Assign a score from 0.0 to 2.0
     based on the change from Output_{k-1} to Output_k. (
     IQS_k below refers to the new 0.0-2.0 scale).
   * **For Output_1 (k=1):**
     * If AC_1 is "Correct": PS_1 = (0.5 + ((IQS_1 - 1.0) /
         2.0)) + 1.0
     * If AC_1 is "Incorrect": PS_1 = IQS_1
   * **For Output_k (k > 1):**
     * If AC_k is "Correct" AND AC_{k-1} was "Incorrect": PS_k
         = [1.7 to 2.0].
     * If AC_k is "Correct" AND AC_{k-1} was "Correct": PS_k =
         [1.1 to 1.3].
     * If AC_k is "Incorrect" AND AC_{k-1} was "Correct": PS_k
         = [0.0 to 0.3].
     * If AC_k is "Incorrect" AND AC_{k-1} was "Incorrect":
         PS_k = [0.7 to 1.2].

You MUST output your evaluation as a JSON list, with one
     object per depth. Each object must contain: "depth_index
     ", "extracted_final_answer", "answer_correctness", "IQS",
      and "PS". Do not include any other text outside the
     JSON list. If the response is a JSON object containing a
      key like "evaluations" which holds the list, please
     ensure the final output is just the list itself.
```

B.3 System prompt — Final-answer Grader C

```
You are an objective AI assistant. You will be given a math
     problem, its ground truth numerical final answer, and a
     single proposed final solution text. Your task is to
     extract the final numerical answer from the proposed
     solution and determine if it matches the ground truth.
     Output a JSON object containing only the key "
     final_correctness_score" with a value of 1.0 if the
     extracted answer matches the ground truth, and 0.0
     otherwise (including if no answer can be reliably
     extracted or if the solution is nonsensical).
```

Usage. During training we call Graders A and B in parallel for every latent depth, average the returned IQS/PS values, then invoke Grader C on the depth-r snapshot to produce the binary correctness reward described in Section 3.2.2 of the main paper.