Self-SoftCoT: A Self-Consistent Framework via Position-Aware Latent Space Reinforcement Learning

Liangliang Dong[1], **Lianlei Shan**[2*], **Shuaimin Li**[3]
[1]College of Software, Jilin University
[2]Tsinghua University
[3]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences
dongll5523@mails.jlu.edu.cn, shanlianlei18@mails.ucas.edu.cn, sm.li2@siat.ac.cn

Abstract

While Chain-of-Thought (CoT) reasoning empowers Large Language Models (LLMs) to tackle complex tasks, its reliance on discrete token decoding imposes an inherent **Discreteness Bottleneck**, limiting expressiveness within a restricted vocabulary space. Existing continuous reasoning approaches, such as SoftCoT (Xu et al., 2025), mitigate this but typically rely on external auxiliary models, resulting in complex deployment and fractured inference pipelines. To address these challenges, we propose **Self-SoftCoT**, a self-contained framework that enables a frozen LLM to internally generate and consume latent thoughts without external assistants. By establishing a single-stream "Thinking $\rightarrow$ Speaking" closed-loop, we decouple latent planning from explicit generation. Furthermore, we adopt **Group Sequence Policy Optimization (GSPO)** to stabilize learning and employ **Position-Aware Independent Projection** to mitigate representation homogenization. Experimental results on five reasoning benchmarks demonstrate that our method significantly improves the reasoning performance of frozen LLMs. Specifically, our Qwen2.5-based model (Yang et al., 2024) uses only $N = 2$ soft tokens to outperform the SoftCoT baseline ($N = 4$), improving the average accuracy from **75.06%** to **78.42%**. Similarly, LLaMA-3.1 (Llama Team, 2024) performance increases from **70.52%** to **74.55%**[1].

1 Introduction

Chain-of-Thought (CoT) prompting has substantially improved the reasoning capabilities of large language models (LLMs) on complex tasks (Wei et al., 2022; Kojima et al., 2022). By explicitly generating intermediate reasoning steps, CoT enables models to decompose multi-hop problems, improve answer accuracy, and provide interpretable reasoning chains. Subsequent works, such as Self-Consistency (Wang et al., 2023) and Tree-of-Thought (Yao et al., 2023), further enhance explicit reasoning trajectories via path sampling and tree search, demonstrating that strategic exploration of reasoning paths can substantially improve both performance and robustness. Collectively, these studies underscore the centrality of intermediate reasoning for tasks ranging from reasoning and decision-making (Shinn et al., 2023) to strategic planning (Yao et al., 2023) and knowledge-intensive question answering (Wei et al., 2022).

Despite these successes, explicit CoT reasoning operates on discrete natural language tokens, introducing a fundamental **Discreteness Bottleneck**. High-dimensional latent thought states must collapse into restricted lexical sequences, which we term an **Expressiveness Bottleneck** (Chen et al., 2025; Hao et al., 2024; Zhu et al., 2025; Xu and Sato, 2025). This limitation prevents LLMs from fully leveraging the high-dimensional latent reasoning space, potentially underutilizing nuanced information in multi-step reasoning tasks. Addressing this challenge requires moving beyond token-level representations toward continuous latent reasoning that can capture richer semantics.

Recent research has explored latent-space reasoning as a solution. Methods such as SoftCoT (Xu et al., 2025) and Coconut (Hao et al., 2024) encode intermediate reasoning in continuous embeddings rather than discrete tokens, enabling more flexible and expressive representations. Coconut alternates between latent and language modes but relies on full-parameter training, posing risks of catastrophic forgetting. SoftCoT demonstrates latent reasoning using externally generated soft tokens, yet its dual-model setup complicates deployment, fragments inference pipelines, and can lead to representation homogenization. These studies highlight both the promise of latent reasoning and the limitations of current approaches, motivating the development of

*Corresponding author.
[1]Our code is available at https://github.com/haha34342/Self-SoftCoT-Code.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 32393–32414
July 2-7, 2026

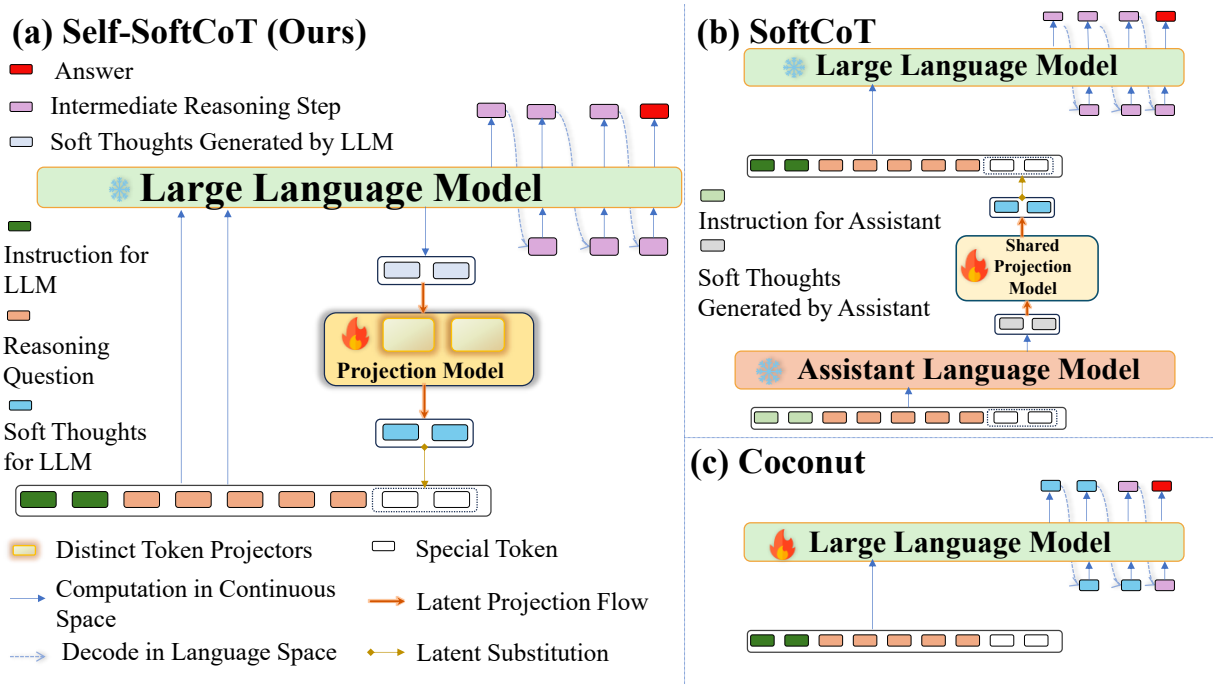

Figure 1: **Architectural Comparison.** (a) **Self-SoftCoT** (Ours) establishes a parameter-efficient single-stream loop via **position-aware independent projection layers** within a frozen LLM. This contrasts with (b) **SoftCoT**, which employs a **shared projection layer** alongside an external assistant model, and (c) **Coconut**, which necessitates full-parameter fine-tuning.

a single-model, self-contained framework.

To further enhance latent reasoning, reinforcement learning (RL) has been applied to optimize latent representations. Hybrid latent reasoning approaches integrate continuous latent states with discrete token signals, using RL to guide richer semantic planning (Yue et al., 2025). Other methods optimize latent reasoning directly without explicit CoT supervision (Zhang et al., 2025), and latent thinking optimization shows how embeddings can encode reward signals for policy updates (Du et al., 2025). However, many existing approaches rely on token-level optimization, which is prone to high gradient variance in long reasoning chains. Methods requiring external assistants or dual-model architectures complicate deployment and may lead to representation homogenization. Moreover, without sequence-level objectives, optimizing latent guidance signals remains unstable and inefficient. These challenges underscore the need for a self-contained latent reasoning framework capable of robust sequence-level optimization within a single model.

Therefore, we propose **Self-SoftCoT**, a self-contained latent reasoning framework in which a frozen LLM internally generates and consumes latent thoughts. Our framework inserts learnable soft token placeholders into the input stream, forming a "Thinking → Speaking" closed-loop entirely within the backbone model, which enables the model to plan and refine its reasoning without external assistants. To preserve expressive and position-specific latent representations, raw latent thoughts are processed via **position-aware independent projection layers** and re-injected into the embeddings. This design simplifies deployment, improves reasoning performance, and helps reduce representation homogenization. In summary, the main contributions of this work are as follows:

- We propose a novel latent reasoning framework, Self-SoftCoT, a single-stream architecture where a frozen LLM internally generates and consumes latent thoughts, eliminating the need for external assistants while preserving rich reasoning capabilities.
- We introduce position-aware independent projection layers for each soft token to mitigate representation homogenization, and adopt a GSPO-style sequence-level objective to optimize these latent projections, improving training stability for our single-model closed-loop architecture under sparse sequence-level rewards.
- Extensive experiments on five benchmarks

(e.g., GSM8K, StrategyQA) demonstrate that our Qwen2.5-based model achieves 78.42% accuracy with only $N = 2$ soft tokens, surpassing SoftCoT ($N = 4$, 75.06%), while LLaMA-3.1 reaches 74.55%, validating the effectiveness and robustness of the proposed framework.

2 Related Work

Chain-of-Thought (CoT) has significantly enhanced LLM reasoning capabilities (Wei et al., 2022; Kojima et al., 2022). Subsequent works like Self-Consistency (Wang et al., 2023) and Tree-of-Thought (Yao et al., 2023) further optimized explicit reasoning trajectories via path sampling and tree search. However, reliance on discrete natural language symbols inevitably introduces a **Discreteness Bottleneck**: forcing high-dimensional dense thought states to collapse into restricted vocabulary sequences constitutes an **Expressiveness Bottleneck**. This discretization limits the model's ability to leverage the high-dimensional properties of the **Latent Reasoning Space** to carry richer semantics.

To overcome this, recent research has turned to latent space reasoning. Coconut (Hao et al., 2024) alternates between language and latent modes but relies on full-parameter training, facing catastrophic forgetting risks. SoftCoT (Xu et al., 2025) validated latent space reasoning via external Assistant-generated soft tokens, but its dual-model architecture (Assistant + Backbone) complicates deployment, fractures the inference pipeline, and suffers from representation homogenization due to shared projections. Self-SoftCoT circumvents external model dependence by constructing a self-consistent "Thinking $\rightarrow$ Speaking" closed-loop within a single frozen model.

In Reinforcement Learning (RL) for LLMs, Proximal Policy Optimization (PPO) (Schulman et al., 2017) combined with RLHF (Ouyang et al., 2022) is the **de facto standard**. However, its reliance on independent Critic networks incurs high memory costs and instability. Group Relative Policy Optimization (GRPO) (Shao et al., 2024) removes the Critic via group advantage normalization, reducing resource demands. Yet, both PPO and GRPO rely on token-level cumulative advantages, making them susceptible to high gradient variance from local probability ratio outliers in multi-step reasoning. In contrast, Group Sequence Policy Optimization (GSPO) (Zheng et al., 2025) elevates optimization to the **sequence level**, utilizing likelihood ratios of the complete answer for importance sampling. This holistic metric smooths local fluctuations, aligning with our hypothesis that latent thoughts provide sequence-level semantic guidance for the output sequence, thus achieving robust updates under final reward signals.

3 Self-SoftCoT Method

This section formalizes the architecture and training objectives of Self-SoftCoT. We propose a single-stream "Thinking $\rightarrow$ Speaking" closed-loop architecture and design a training strategy balancing exploration and robustness based on GSPO sequence-level optimization.

3.1 Problem Setup and Notation

Given a natural language question x, the model outputs answer y. We introduce N special soft token slots at the input side of the backbone LLM to carry latent thoughts. We define the backbone LLM as a mapping function from input sequence space to output sequence space:

$$\text{LLM}_{\Theta_0} : \underbrace{\mathcal{V}^*}_{\text{Input Space}} \rightarrow \underbrace{\mathcal{V}^*}_{\text{Output Space}}, \quad (1)$$

where $\mathcal{V}$ is the vocabulary set, and $\mathcal{V}^*$ denotes the set of finite sequences of arbitrary length. Θ_0 represents pre-trained parameters, which remain **fully frozen** during training.

We introduce position-aware projection parameters $\theta = \{(W_i, b_i)\}_{i=1}^{N}$, where $W_i \in \mathbb{R}^{d\times d}, b_i \in \mathbb{R}^d$. These parameters constitute the latent thought layer and are the only trainable components.

3.2 Decoupled Architecture

To construct a self-consistent reasoning loop within a single model, we formalize the process into three tightly coupled stages: **Thinking**, **Projecting**, and **Hybrid Speaking**. This architecture enables the model to perform internal latent planning before explicit generation via dynamic modification of the embedding space.

3.2.1 Stage 1: Latent Thought Generation

First, we construct an input sequence x containing N thought placeholders. Let $E(\cdot)$ be the model's embedding layer. We feed the sequence into the frozen LLM_{Θ_0} to initiate the **Thinking Phase**, extracting the last-layer hidden states at placeholder positions $t_1, \ldots, t_N$:

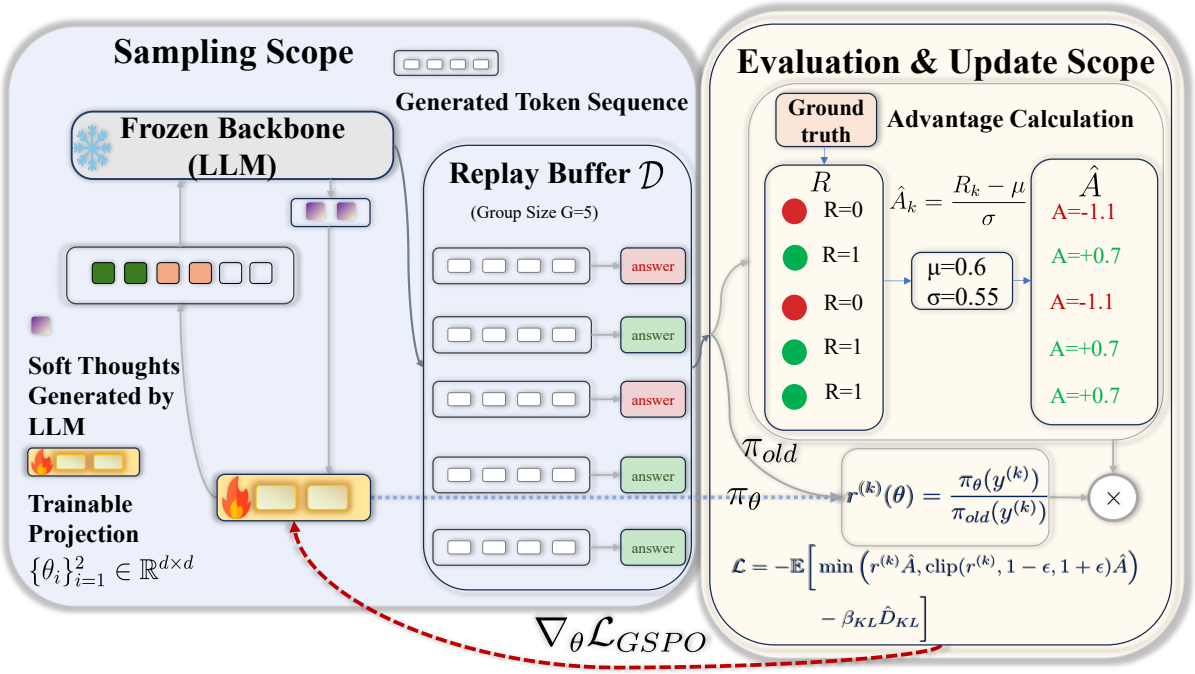

Figure 2: **GSPO Training Workflow.** Illustrating the Sampling Scope, Replay Buffer, and Evaluation Scope. R denotes the raw binary reward, and $\hat{A}$ represents the advantage calculated via group normalization, **where μ and σ denote the mean and unbiased sample standard deviation, respectively.** k is the trajectory index, π_{old} is the cached behavior policy in $\mathcal{D}$, and π_θ is the current policy.

$$z_i = \text{LLM}_{\Theta_0}(E(x))_{t_i} \in \mathbb{R}^d, \quad i = 1, \ldots, N. \tag{2}$$

The set $\mathcal{Z} = \{z_i\}_{i=1}^N$ represents the "raw thought stream" spontaneously generated by the model based on the current context.

3.2.2 Stage 2: Independent Projection

To transform raw hidden states into latent guidance signals comprehensible to the input layer, we introduce position-aware independent projection transformations ϕ. For the i-th thought slot, the projected **Soft Thought Vector** is calculated as:

$$\tilde{z}_i = \phi_i(z_i; \theta) = W_i \cdot z_i + b_i. \tag{3}$$

Zero Initialization Strategy. In our robust default recipe (Config B), we initialize W_i, b_i to zero, such that initially $\tilde{z}_i = \mathbf{0}$. Although this eliminates **semantic noise** from random initialization, the insertion of placeholders itself introduces a **Positional Shift**, constituting a slight structural perturbation (see analysis in Section 5.1).

3.2.3 Stage 3: Hybrid Speaking

This is the core step. Instead of decoding $\tilde{z}_i$ into discrete text, we directly "re-inject" it into the input stream via an **Embedding Replacement** mechanism, as illustrated in Figure 1.

Specifically, we construct a **Hybrid Embedding Sequence $\mathbf{H}_x$**, retaining original word embeddings $E(x_j)$ at normal token positions while forcing replacement with projected soft thought vectors $\tilde{z}_i$ at thought slot positions t_i:

$$\mathbf{H}_x[j] = \begin{cases} \tilde{z}_k & \text{if } j = t_k \text{ for some } k \text{ (Thought Slot)} \\ E(x_j) & \text{otherwise (Text Token)} \end{cases} \tag{4}$$

During the **Speaking Phase**, the model generates the final answer y based on this hybrid embedding. The conditional probability distribution is conditioned on the hybrid embedding sequence, which includes the re-injected latent thought stream:

$$p_{\Theta_0,\theta}(y \mid x) = \prod_{k=1}^{|y|} p_{\Theta_0}\left(y_k \mid \text{Concat}(\mathbf{H}_x, E(y_{<k}))\right) \tag{5}$$

This mechanism establishes a structured "Thinking $\rightarrow$ Speaking" closed-loop consisting of two phases. In **Phase 1 (Thinking)**, the frozen model processes the input prefix to produce raw thought states $\mathcal{Z}$. Subsequently, in **Phase 2 (Speaking)**, the projected $\tilde{\mathcal{Z}}$ is re-injected to replace original placeholders, providing sequence-level semantic guidance for the model to autoregressively generate the final response y.

3.3 Reward and Advantage

We define the binary reward $R(x, \hat{y}) \in \{0, 1\}$ based on the final answer correctness. As illustrated in Figure 2, we employ a Group Advantage Estimation strategy to normalize these raw rewards. Specifically, for a sampled group of G trajectories $\{\hat{y}^{(k)}\}_{k=1}^{G}$, we compute the advantage $\hat{A}^{(k)}$ as:

$$\hat{A}^{(k)} = \frac{R^{(k)} - \mu_{\text{group}}}{\sigma_{\text{group}} + \epsilon}, \tag{6}$$

where $R^{(k)}$ is the reward of the k-th trajectory, and $\mu_{\text{group}}, \sigma_{\text{group}}$ denote the mean and **unbiased sample standard deviation** within the group.

3.4 The GSPO Algorithm

To mitigate the high variance inherent in token-level feedback during multi-step reasoning, we adopt the GSPO algorithm (Zheng et al., 2025). While GRPO (Shao et al., 2024) reduces memory overhead by discarding the Critic, it retains **token-level importance weighting**, which often yields unstable gradient estimates for long sequences where early latent decisions impact global outcomes. In contrast, GSPO leverages a **length-normalized sequence-level likelihood ratio** for importance sampling. This mechanism smooths local fluctuations, improving robust optimization for multi-step reasoning tasks under sparse rewards.

3.4.1 Sequence-Level Importance Sampling

For an input question x, we sample a group of trajectories $\mathcal{G} = \{\hat{y}^{(k)}\}_{k=1}^{G}$ from the behavior policy $\pi_{\theta_{\text{beh}}}$ (stored in the Replay Buffer $\mathcal{D}$ shown in Figure 2). The overall training workflow is depicted in Figure 2. Let $\Delta\ell_t^{(k)}$ denote the log-probability difference between the current and behavior policies at step t. The sequence-level importance ratio $r^{(k)}(\theta)$ is formalized as the geometric mean of the likelihood ratios over valid tokens:

$$r^{(k)}(\theta) = \exp\left(\frac{\sum_{t=1}^{T} m_t^{(k)} \cdot \Delta\ell_t^{(k)}}{\sum_{t=1}^{T} m_t^{(k)} + \delta}\right), \tag{7}$$

where $m_t^{(k)}$ is the validity mask and δ ensures numerical stability. The denominator performs **length normalization**, preventing the ratio scale from diverging with generation length T and maintaining consistent update steps across trajectories of varying lengths.

3.4.2 Optimization Objective

To constrain policy drift while maximizing the group-normalized advantage $\hat{A}^{(k)}$, we employ a composite objective incorporating asymmetric clipping and KL regularization. The objective is defined as:

$$\mathcal{L}^{\text{clip}}(\theta) = \min\Big(r^{(k)}(\theta)\hat{A}^{(k)}, \text{clip}\big(r^{(k)}(\theta), 1-\epsilon_l, 1+\epsilon_r\big)\hat{A}^{(k)}\Big), \tag{8}$$

$$\mathcal{L}(\theta) = \mathbb{E}_{\mathcal{G}}\left[-\mathcal{L}^{\text{clip}}(\theta) + \beta_{\text{KL}} \cdot \widehat{\mathcal{D}}_{\text{KL}}\right]. \tag{9}$$

Here, $\mathcal{L}^{\text{clip}}$ applies asymmetric constraints to the surrogate objective. $\widehat{\mathcal{D}}_{\text{KL}}$ represents the KL divergence penalty, which we approximate via a second-order token-level Mean Squared Error (MSE) to ensure computational efficiency (see **Appendix B.1** for the detailed derivation and hyperparameter settings).

3.5 Training Recipes

Different backbone models exhibit varying sensitivities to latent space perturbations. To address this, we introduce two distinct training paradigms: **Exploratory Streaming** for maximizing performance upper bounds on robust models, and **Robust Experience Replay** for stabilizing initialization-sensitive models. The specific configurations and hyperparameter settings for these recipes are detailed in Section 4.3.

4 Experimental Setup

This section outlines the evaluation benchmarks, comparative baselines, and the implementation details of our training recipes.

4.1 Benchmarks and Datasets

We conduct experiments on five reasoning datasets spanning three categories: **mathematical reasoning**, **commonsense reasoning**, and **symbolic reasoning**. For mathematical reasoning, we utilize GSM8K (Cobbe et al., 2021), ASDiv-Aug (Xu et al., 2025), and AQuA (Ling et al., 2017). For commonsense reasoning, we employ StrategyQA (Geva et al., 2021). For symbolic reasoning, we use Date Understanding (Srivastava et al., 2023). Detailed statistics are provided in Appendix C.1.

4.2 Baselines

We compare Self-SoftCoT against four representative baselines: (1) **Zero-Shot CoT** (Standard baseline); (2) **LoRA Fine-tuning** (Parameter-efficient

fine-tuning baseline); (3) **Coconut** (Hao et al., 2024) (Latent reasoning baseline); (4) **SoftCoT** (Xu et al., 2025) (Representative continuous reasoning baseline). Detailed definitions and settings are provided in Appendix C.2.

4.3 Implementation Details

We conducted experiments on **Qwen2.5-7B-Instruct** (Yang et al., 2024) and **LLaMA-3.1-8B-Instruct** (Llama Team, 2024). We distinguish **literature-reported baselines** from **our newly conducted runs**. Specifically, **Zero-Shot CoT**, **LoRA Fine-tuning**, **Coconut**, and **SoftCoT** are directly reported from Xu et al. (2025) for comparison. In contrast, the $N = 0$ **Baseline**, $N = 2$ **(Untrained)**, **Config A**, and **Config B** are our own runs. The $N = 0$ **Baseline** is implemented using the corresponding task template in Appendix I, with placeholders removed to ensure fair comparison.

To balance exploration and robustness, we implemented two recipes introduced in Section 3.5:

(1) Config A (Exploratory): Designed to explore higher performance potentials, this configuration employs **Identity Mapping Initialization** and an **Online Streaming** update strategy. It adopts relaxed regularization ($\beta_{\text{KL}} = 0$) to encourage broad exploration, primarily applied to robust backbones like Qwen2.5 to investigate enhanced reasoning capabilities.

(2) Config B (Robust): Adopted as our standardized setting, this configuration yields strong results on both **Qwen2.5** and **LLaMA-3.1**, making it a versatile choice for different backbones. It employs **Zero Initialization** to effectively minimize initial cold-start perturbations, utilizes an **Experience Replay Buffer** to smooth out gradient variance, and applies strict KL regularization ($\beta_{\text{KL}} = 0.01$) to prevent excessive policy drift, ultimately ensuring consistent and stable convergence across diverse reasoning tasks.

Detailed hyperparameters are provided in Appendix D.2.

5 Results and Analysis

This section reports the main results of Self-SoftCoT on **Qwen2.5-7B-Instruct** and **LLaMA-3.1-8B-Instruct**, analyzing generalization behavior and internal parameter update divergence.

5.1 Main Results

Tables 1 and 2 present the performance of **Qwen2.5-7B-Instruct** and **LLaMA-3.1-8B-Instruct after training**.

The results indicate that Self-SoftCoT achieves significant performance gains across both backbone models. For **Qwen2.5**, both the ceiling-oriented Config A (78.70%) and the stability-oriented Config B (78.42%) surpass SoftCoT ($N = 4$) at 75.06%. Config B, despite introducing KL constraints, maintains strong competitiveness on AQuA and StrategyQA, demonstrating the efficacy of Buffer Replay on these reasoning tasks.

For **Qwen2.5** and **LLaMA-3.1**, the performance of the $N = 2$ Zero Initialization (untrained) variants (69.96% and 67.23%, respectively) is slightly lower than the $N = 0$ baselines (71.17% and 68.84%). This indicates that even with zero-vector injection, the insertion of extra token slots causes structural perturbations via sequence length changes and **Positional Shift**. However, after training with Config B, LLaMA-3.1's performance rebounds strongly to **74.55%**, significantly outperforming SoftCoT's 70.52%. This suggests that the synergistic strategy of **Zero Initialization, GSPO, and Buffer Replay** not only overcomes structural interference during cold start but also effectively improves latent space reasoning capabilities while preserving backbone knowledge.

Furthermore, significant gains are achieved under the zero initialization of our standardized Config B. Given that the projection layer is small relative to the frozen backbone (approximately $2(d_{\text{model}}^2 + d_{\text{model}})$ trainable parameters for $N = 2$, i.e., about $2d_{\text{model}}^2$ for large d_{model}), and that the post-training **Parameter Shift** remains quantitatively small (e.g., Frobenius Norm < 0.6 on **LLaMA-3.1-8B-Instruct** across the four analyzed tasks, as shown in Table 3), the model can effectively optimize reasoning trajectories with only limited weight updates. This supports the **Parameter Efficiency** of the projection layer as a lightweight "Semantic Bridge". It also suggests that the method's performance potential may not yet be fully exhausted, and could be further explored with larger data scales or extended training schedules.

5.2 Analysis: Parameter Update Divergence

To investigate whether independent projection layers exhibit differentiated adaptation patterns, we

Model / Method	GSM8K	ASDiv-Aug Math Reasoning	AQuA	StrategyQA Commonsense	DU Symbolic	Avg
Zero-Shot CoT	83.70 ± 0.78	87.19 ± 0.28	64.53 ± 3.27	49.65 ± 3.18	66.40 ± 2.26	70.29
LoRA Fine-Tuning	81.80 ± 0.00	86.80 ± 0.00	62.60 ± 0.00	-	-	-
Coconut (Hao et al., 2024)	82.49 ± 0.00	86.90 ± 0.00	63.39 ± 0.00	-	-	-
SoftCoT ($N = 4$)	85.81 ± 1.82	88.90 ± 1.01	72.44 ± 2.19	60.61 ± 1.55	67.52 ± 2.92	75.06
Self-SoftCoT Variants (Ours)						
$N = 0$ Baseline	84.18 ± 1.60	87.15 ± 1.01	67.72 ± 3.96	52.88 ± 5.07	63.92 ± 5.80	71.17
$N = 2$ (Untrained)	81.87 ± 3.70	86.72 ± 3.19	66.45 ± 2.77	52.62 ± 7.28	62.16 ± 2.17	69.96
Config A (Stream, Explore)	87.97 ± 0.79	$\mathbf{91.21 \pm 0.56}$	77.56 ± 0.48	66.29 ± 0.91	$\mathbf{70.48 \pm 1.51}$	**78.70**
Config B (Buffer, Robust)	$\mathbf{88.89 \pm 0.59}$	90.48 ± 0.40	$\mathbf{78.11 \pm 0.91}$	$\mathbf{66.94 \pm 1.30}$	67.68 ± 1.15	78.42

Table 1: Qwen2.5-7B-Instruct Results. Config B, adopted as our standardized recipe, achieves robust performance gains and significantly outperforms SoftCoT, LoRA, and Coconut.

Model / Method	GSM8K	ASDiv-Aug Math Reasoning	AQuA	StrategyQA Commonsense	DU Symbolic	Avg
Zero-Shot CoT	79.61 ± 0.81	86.78 ± 0.63	54.65 ± 2.43	65.63 ± 3.51	54.40 ± 2.40	68.21
LoRA Fine-Tuning	75.66 ± 0.00	86.67 ± 0.00	52.36 ± 0.00	-	-	-
Coconut (Hao et al., 2024)	76.12 ± 2.00	86.80 ± 1.00	53.12 ± 0.00	-	-	-
SoftCoT ($N = 4$)	81.03 ± 0.42	87.19 ± 0.40	56.30 ± 1.67	69.04 ± 1.23	59.04 ± 1.93	70.52
Self-SoftCoT Variants (Ours)						
$N = 0$ Baseline	79.76 ± 0.73	88.11 ± 0.81	55.36 ± 1.73	65.46 ± 2.49	55.52 ± 1.25	68.84
$N = 2$ (Untrained)	77.80 ± 1.36	87.48 ± 0.76	53.07 ± 1.63	65.02 ± 3.18	52.80 ± 1.39	67.23
Config B (Buffer, Robust)	$\mathbf{82.91 \pm 1.06}$	$\mathbf{92.35 \pm 1.54}$	$\mathbf{63.31 \pm 2.88}$	$\mathbf{69.30 \pm 1.14}$	$\mathbf{64.88 \pm 1.73}$	**74.55**

Table 2: LLaMA-3.1-8B-Instruct Results. Our robust training recipe effectively activates latent reasoning, surpassing standard CoT and SoftCoT.

Task Name	Token 0 Shift	Token 1 Shift	Tok 0 vs 1 Sim
StrategyQA	0.5430	0.5430	0.3359
GSM8K	0.4492	0.3730	0.0474
AQuA	0.3809	0.5039	0.0359
ASDiv	0.1543	0.1836	0.1069

Table 3: Parameter update characteristics of projection layers on LLaMA-3.1. Token shift reflects learning intensity; similarity reflects directional differentiation of parameter updates.

Task Pair	Token 0 Sim	Token 1 Sim
GSM8K vs AQuA	0.0339	0.0116
GSM8K vs StrategyQA	0.0075	0.0035
StrategyQA vs AQuA	0.0124	0.0043

Table 4: Cross-task projection layer update similarity. Low similarity indicates task-specific adaptation.

analyzed the projection layer configurations of **LLaMA-3.1-8B-Instruct**. We flattened the high-dimensional projection matrices into 1D vectors and calculated two key metrics. We primarily compute the **Frobenius Norm (Parameter Shift)** $\|\mathbf{W}\|_F$ to reflect the magnitude of parameter updates from zero initialization. Simultaneously, we calculate the **Cosine Similarity** $\cos(\theta)$ to measure the directional consistency of parameter updates between different token slots.

As shown in Table 3, on logic-intensive math tasks like GSM8K and AQuA, the parameter update similarity between Token 0 and Token 1 is extremely low (0.0474 and 0.0359, respectively), indicating that the two projection layers spontaneously evolved **Distinct Update Directions**. This suggests that independent projection layers facilitate partially complementary update patterns in latent thought during complex reasoning, thereby helping mitigate representation homogenization often associated with single projection layers. Conversely, similarity is higher on StrategyQA (0.3359). We attribute this to the relatively short reasoning chains and simpler semantics of binary classification (Yes/No), which reduce the demand for heterogeneous latent information and cause update directions to converge. This is consistent with the adaptivity of Self-SoftCoT: more differentiated updates for complex logical tasks and more convergent updates for simpler tasks.

Furthermore, cross-task similarity analysis (Table 4) reveals that updates for the same token position vary significantly across tasks (e.g., GSM8K vs. AQuA is only 0.0339). This indicates that projection layers adapt flexibly to specific task semantics rather than learning generic heuristic patterns.

GSM8K Configuration	Initial	Trained
$N = 0$ (Baseline)	84.18 ± 1.60	-
$N = 1$	82.79 ± 3.04	87.63 ± 1.51
$N = 2$ (**Ours**)	81.87 ± 3.70	$\mathbf{88.89 \pm 0.59}$
w/ Shared Proj.	81.87 ± 3.70	87.90 ± 0.61
w/ Dual (1.5B Asst.)	81.87 ± 3.70	87.08 ± 0.42
$N = 3$	81.59 ± 3.41	88.04 ± 0.86
$N = 4$	81.40 ± 4.31	87.96 ± 0.35

StrategyQA Config	Initial	Trained
$N = 0$ (Baseline)	52.88 ± 5.07	-
$N = 1$	53.01 ± 4.70	-
$N = 2$ (**Ours**)	52.62 ± 7.28	$\mathbf{66.94 \pm 1.30}$
w/ Shared Proj.	52.62 ± 7.28	65.20 ± 1.82
w/ Dual (1.5B Asst.)	52.62 ± 7.28	-
$N = 3$	51.05 ± 9.18	-
$N = 4$	49.56 ± 9.87	-

Table 5: Ablation study on soft token count (N) and projection architectures (Mean $\pm$ Std, 5 seeds). **Note:** We aligned Dual Model prompts with Self-SoftCoT for strict control. Thus, its initial performance matches the $N = 2$ untrained baseline due to **zero initialization**.

5.3 Ablation Studies

Self-SoftCoT consistently employs $N = 2$ across all tasks. To validate this hyperparameter and isolate the contributions of the Position-Aware Independent Projection architecture, we conducted ablation studies on GSM8K (Arithmetic) and StrategyQA (Commonsense). The results are presented in Table 5.

5.3.1 Ablation on Token Count N

As shown in Table 5, a scan of $N \in \{0, \ldots, 4\}$ on GSM8K justifies the choice of $N = 2$. First, regarding the **structural perturbation during initialization**, observing the Initial Baseline column reveals that introducing untrained tokens consistently lowers performance below the $N = 0$ baseline. The overall trend confirms that extra slots introduce structural perturbation via sequence length changes and **Positional Shift**. $N = 2$, despite some perturbation (81.87%), retains high initial performance, constituting an optimal trade-off between latent space capacity and initialization stability, providing an ideal cold start for fine-tuning.

Second, regarding the **post-training performance trade-off**, after GSPO training, performance peaks at $N = 2$ (88.89%). Notably, the high standard deviation in the 'Initial' column reflects initialization instability, while the significantly reduced variance in the 'Trained' column (e.g., Std drops from 3.70 to 0.59 on GSM8K) demonstrates that our GSPO training successfully recovers the model to achieve robust convergence. However, compared to $N \geq 3$, performance degrades despite the theoretical increase in capacity. We attribute this phenomenon to two synergistic factors: **Input Embedding Distribution Shift**, where excessive soft tokens may cause the combined input embeddings to deviate significantly from the pre-training manifold; and **Optimization Challenges**, where increasing N expands the search space, leading to insufficient exploration **under our current training strategy**. Thus, $N = 2$ represents the optimal equilibrium point between latent capacity, distributional compatibility, and training efficiency.

5.3.2 Efficacy of Independent Projection

To verify the superiority of "Position-Aware Independent Projection" over traditional shared projection, we controlled variables with fixed $N = 2$. Concerning the significance of performance gains, independent projection outperforms in both task types, with gains of +0.99% on GSM8K and +1.74% on StrategyQA, suggesting the generalization benefit of this design. Furthermore, shared projection forces all thought slots to interact via the same linear transformation, which can lead to homogenization. In contrast, independent projection allows the model to allocate position-specific parameter spaces to different slots, successfully allowing tokens to develop distinct adaptation patterns. Finally, comparing single-stream efficacy against dual-model redundancy, we found that a dual-model variant (using a 1.5B assistant) underperformed our framework (87.08% vs. 88.89%). We attribute this to the **feature space misalignment** between heterogeneous models, whereas our single-stream architecture operates within a unified **iso-dimensional latent space**, ensuring that latent thoughts naturally align with the representation manifold of the backbone.

6 Conclusion

In this paper, we introduce **Self-SoftCoT**, a self-contained latent space reasoning framework that eliminates the need for external auxiliary models. By establishing a "Thinking $\rightarrow$ Speaking" closed-loop within a single frozen backbone, our method structurally decouples thought generation from consumption. We introduce position-aware independent projection modules to mitigate representation homogenization and support position-specific adaptation. To ensure robust optimization, we propose

a standardized training recipe based on Group Sequence Policy Optimization (GSPO), which utilizes sequence-level importance sampling and experience replay to mitigate high variance in multi-step reasoning. Experiments on five benchmarks demonstrate that our framework significantly outperforms explicit CoT and existing latent space methods on Qwen2.5 and LLaMA-3.1, maintaining high deployment simplicity. Future work will extend this efficient architecture to dynamic soft token mechanisms and multimodal reasoning scenarios.

Limitations

While Self-SoftCoT effectively balances deployment simplicity and reasoning performance, several limitations remain that warrant future investigation.

First, the **Interpretability Trade-off** persists. Although the framework ultimately decodes its final responses in natural language, its internal planning and intermediate reasoning are compressed into continuous soft tokens. These latent representations lack the semantic transparency of explicit text. While this is a deliberate design choice, it sacrifices the human-readable reasoning steps that are crucial for debugging and trust. Future work should explore hybrid approaches that interleave explicit textual reasoning with continuous latent states, aiming to achieve an optimal balance between computational efficiency and process interpretability.

Second, the **Static Soft Token Budget** (e.g., fixing $N = 2$) prevents the dynamic allocation of latent computation based on problem difficulty. Although the model can still generate variable-length explicit text during the speaking phase, its internal planning capacity is strictly bounded by the fixed number of soft placeholders. This static design limits the framework's adaptivity, potentially constraining its ability to encode sufficient intermediate states for exceptionally complex problems that might require a larger, variable soft token budget.

Third, relying solely on **Final Answer Correctness** as a sparse reward signal can occasionally lead to semantic drift in intermediate latent representations during optimization. Furthermore, on exceptionally simple or excessively difficult problems where the reward variance approaches zero, advantage estimation becomes challenging. Although our replay buffer and zero-variance skipping mechanism effectively mitigate this instability in practice, incorporating dense, step-level process rewards could further stabilize the optimization of latent guidance signals. In addition, the latent nature of the reasoning process may introduce practical reliability and deployment risks. Because intermediate reasoning is encoded in continuous representations rather than explicit text, errors in planning may be harder to inspect, diagnose, or intervene on, even when the final answer appears fluent and confident. This limitation may reduce auditability in high-stakes settings and suggests that Self-SoftCoT should be deployed with caution in scenarios requiring transparent human oversight. Finally, while our ablation studies suggest a degree of zero-shot cross-domain generalization of the projection layers, the framework was primarily evaluated under a **Task-Specific Training** paradigm. Developing a unified, universal latent projection module capable of handling omni-domain reasoning without specialized fine-tuning remains an important objective for future research.

Acknowledgments

The first author sincerely thanks his supervisor, Lianlei Shan, for his guidance, encouragement, and valuable feedback throughout this research. He also thanks Jilin University and the College of Software for their support and for providing a strong academic environment. Finally, he is grateful to his parents for their constant support and encouragement. The authors also used generative AI tools for limited language assistance during manuscript preparation, including minor wording refinement and polishing of the authors' original text. All technical content, experimental design, results, and final writing decisions were verified and determined by the authors.

References

Xinghao Chen, Anhao Zhao, Heming Xia, Xuan Lu, Hanlin Wang, Yanjun Chen, Wei Zhang, Jian Wang, Wenjie Li, and Xiaoyu Shen. 2025. Reasoning beyond language: A comprehensive survey on latent chain-of-thought reasoning. *CoRR*, abs/2505.16782.

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. *CoRR*, abs/2110.14168.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing rea-

soning capability in llms via reinforcement learning. *CoRR*, abs/2501.12948.

Hanwen Du, Yuxin Dong, and Xia Ning. 2025. Latent thinking optimization: Your latent reasoning language model secretly encodes reward signals in its latent thoughts. *CoRR*, abs/2509.26314.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. 2024. KTO: model alignment as prospect theoretic optimization. *CoRR*, abs/2402.01306.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? A question answering benchmark with implicit reasoning strategies. *Trans. Assoc. Comput. Linguistics*, 9:346–361.

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

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

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

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 158–167. Association for Computational Linguistics.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021. GPT understands, too. *CoRR*, abs/2103.10385.

Llama Team. 2024. The llama 3 herd of models. *CoRR*, abs/2407.21783.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Qwen Team. 2025. Qwen3 technical report. *CoRR*, abs/2505.09388.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, Agnieszka Kluska, Aitor Lewkowycz, Akshat Agarwal, Alethea Power, Alex Ray, Alex Warstadt, Alexander W. Kocurek, Ali Safaya, Ali Tazarv, and 431 others. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Trans. Mach. Learn. Res.*, 2023.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Kevin Xu and Issei Sato. 2025. A formal comparison between chain-of-thought and latent thought. *CoRR*, abs/2509.25239.

Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. 2025. Softcot: Soft chain-of-thought for efficient reasoning with llms. In *Proceedings of the 63rd Annual Meeting of the Association for Computational*

Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025, pages 23336–23351. Association for Computational Linguistics.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024. Qwen2.5 technical report. *CoRR*, abs/2412.15115.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Zhenrui Yue, Bowen Jin, Huimin Zeng, Honglei Zhuang, Zhen Qin, Jinsung Yoon, Lanyu Shang, Jiawei Han, and Dong Wang. 2025. Hybrid latent reasoning via reinforcement learning. *CoRR*, abs/2505.18454.

Xiaotian Zhang, Chunyang Li, Yi Zong, Zhengyu Ying, Liang He, and Xipeng Qiu. 2023. Evaluating the performance of large language models on GAOKAO benchmark. *CoRR*, abs/2305.12474.

Yang Zhang, Wenxin Xu, Xiaoyan Zhao, Wenjie Wang, Fuli Feng, Xiangnan He, and Tat-Seng Chua. 2025. Reinforced latent reasoning for llm-based recommendation. *CoRR*, abs/2505.19092.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. 2025. Group sequence policy optimization. *CoRR*, abs/2507.18071.

Ruijie Zhu, Tianhao Peng, Tianhao Cheng, Xingwei Qu, Jinfa Huang, Dawei Zhu, Hao Wang, Kaiwen Xue, Xuanliang Zhang, Yong Shan, Tianle Cai, Taylor Kergan, Assel Kembay, Andrew Smith, Chenghua Lin, Binh Nguyen, Yuqi Pan, Yuhong Chou, Zefan Cai, and 14 others. 2025. A survey on latent reasoning. *CoRR*, abs/2507.06203.

A Overview of Appendices

This appendix provides supplementary materials to support the reproducibility and detailed analysis of **Self-SoftCoT**. The structure is organized as follows:

- **Appendix B**: Methodological derivations, including the KL divergence approximation (Supplementary to Section 3).
- **Appendix C**: Experimental setup details, including dataset statistics and baseline definitions (Supplementary to Section 4).
- **Appendix D**: Comprehensive implementation details, training recipes, hyperparameters, and reproduction commands (Supplementary to Section 4.3).
- **Appendix E**: Additional evaluations on multi-step reasoning benchmarks (MATH-500 (Lightman et al., 2023), Gaokao2023 (Zhang et al., 2023)) and cross-domain zero-shot generalization.
- **Appendix F**: Inference latency and efficiency comparisons against explicit long-chain reasoning models (e.g., DeepSeek-R1).
- **Appendix G**: Ablation studies on reinforcement learning algorithms (DPO, KTO, GRPO vs. GSPO) and analysis of policy collapse.
- **Appendix H**: Disentanglement of performance gains, validating the necessity of dynamic latent thoughts over static parameters.
- **Appendix I**: Full prompt templates for both Self-SoftCoT and the SoftCoT baseline, detailing the specific instruction formats and special token handling for fair comparison.
- **Appendix J**: Qualitative input-output case studies comparing our method with standard CoT and SoftCoT.

B Methodological Details

B.1 KL Divergence Approximation

In Section 3.4, we employ a KL divergence penalty $\widehat{\mathcal{D}}_{\mathrm{KL}}$ to regularize the policy update. To enhance computational efficiency and smooth gradients, we utilize the second-order approximation based on token-level Mean Squared Error (MSE), following Schulman et al. (2017):

$$\widehat{\mathcal{D}}_{\mathrm{KL}} \approx \frac{1}{2} \cdot \frac{\sum_{t=1}^{T} m_t^{(k)} (\Delta \ell_t^{(k)})^2}{\sum_{t=1}^{T} m_t^{(k)} + \delta}. \tag{10}$$

This approximation maintains numerical stability within the small trust region where $\theta \approx \theta_{\text{beh}}^{(k)}$, which is guaranteed by our strict clipping thresholds.

C Experimental Setup Details

C.1 Datasets and Benchmarks

The detailed statistics of the five benchmarks used in our experiments are listed in Table 6.

Task	Train	Val†	Test	Domain
GSM8K	7,000	150	1,319	Math Word Problems
ASDiv-Aug	4,000	150	1,038	Arithmetic Augmented
AQuA	97,467	254	254	Algebra (MCQ)
StrategyQA	1,603	229	458	Commonsense
Date Understanding	-	-	250	Symbolic Reasoning

Table 6: Detailed statistics of the datasets used in evaluation.

C.2 Baseline Details

We provide detailed definitions for the baselines used in our experiments. For fair comparison, we cite results from Xu et al. (2025), while strictly reproducing the standard CoT baseline in our configuration.

Zero-Shot CoT. The standard Chain-of-Thought performance reported by Xu et al. (2025). We use this reported value as a reference anchor for the other baselines cited from the same literature.

LoRA Fine-tuning. We reference the performance of Low-Rank Adaptation (LoRA) reported by Xu et al. (2025). This serves as a standard parameter-efficient fine-tuning baseline for comparison.

Coconut. A continuous latent reasoning method (Hao et al., 2024). We cite the performance metrics reported in Xu et al. (2025) for direct comparison with a representative latent reasoning baseline.

SoftCoT. A representative continuous reasoning method ($N = 4$) proposed by Xu et al. (2025) (Representative continuous reasoning baseline).

$N = 0$ **Baseline.** We reproduced the Zero-Shot CoT setting using the same configuration as our

Component	Config A (Exploratory)	Config B (Robust)
Backbone	Qwen2.5	LLaMA-3.1 / Qwen2.5
Initialization	Identity ($W = I$)	Zero ($W = 0$)
Optimization	Online Streaming	Experience Replay (Buffer)
KL Reg	None ($\beta = 0$)	Yes ($\beta = 0.01$)

Table 7: Comparison of Training Recipes. Config B is the recommended standardized setting.

proposed method. This serves as the primary control group for calculating the relative gains of our approach.

$N = 2$ **(Untrained).** Our control group to assess the initial structural perturbation impact of inserting initialized soft token slots before training.

D Implementation Details

This section provides exhaustive experimental configurations, hyperparameter explanations, and core training commands to ensure reproducibility. We adopt a **Task-Specific Training** paradigm: a set of projection layer parameters is trained independently for each dataset. All experiments were conducted on NVIDIA RTX 5090 GPUs.

D.1 Training Recipes

D.2 Environment and Hyperparameters

We employed the **Config B: Robust Recipe** defined in Section 3.5 as the standard setting. This recipe ensures cross-model training stability via zero initialization and experience replay.

Table 8 details the specific parameter instantiation for Qwen2.5 and LLaMA-3.1 under this recipe. While maintaining a **standardized** training strategy, we adapted hyperparameters such as learning rate based on backbone sensitivity.

Rationale for GSPO Hyperparameters:

- **Asymmetric Tight Clipping ($\epsilon_l = 3$e-$4, \epsilon_r = 4$e-$4):** This threshold is significantly lower than traditional PPO (typically 0.2). Although length normalization ensures numerical stability of the ratio, the core motivation is to **strictly suppress Excessive Updates in long-sequence high-dimensional space**. Under the sequence-level mean metric, the sequence ratio $r(\theta)$ is extremely sensitive to policy changes: even minute drifts in average log-likelihood (e.g., $\Delta \approx 3 \times 10^{-4}$) imply significant deviation in the policy distribution over a long chain. Therefore, a very narrow Trust Region must be imposed to prevent Policy Collapse. The asymmetric design ($\epsilon_r > \epsilon_l$) aims to grant slightly more update freedom to positive samples under strict constraints.
- **Numerical Stability Constant ($\delta = 1$e-8):** Used in the denominator term `mask.sum() + δ`, this aims to eliminate division-by-zero risks caused by extremely short or fully masked sequences, ensuring numerical robustness in large-batch training.

Hyperparameter	Qwen2.5-7B-Instruct	LLaMA-3.1-8B-Instruct
Training Strategy (Robust Recipe / Config B)		
Init Strategy	**Zero initialization** ($W = 0$)	**Zero initialization** ($W = 0$)
RL Policy	**Experience replay (Buffer)**	**Experience replay (Buffer)**
Update Epochs	3 (Multi-Epoch)	3 (Multi-Epoch)
Buffer Strategy	Clear after each round	Clear after each round
Optimization (Model-Specific)		
Learning Rate	1×10^{-5}	5×10^{-6}
Optimizer	AdamW	AdamW
Batch Size	5	5
Group Size (G)	5	5
Gradient Clipping	1.0	1.0
GSPO Specifics		
ϵ (Clip Range)	[3e-4, 4e-4] (Asymmetric)	
β_{KL} (KL Coeff)	0.01	
δ (Stability)	1e-8	
Resources		
Max Gen Length	1024	1024
Total Steps	150	150
Episodes/Round	16	16

Table 8: Detailed training hyperparameters based on **Config B (Robust Recipe)**. While the training strategy remains consistent (Zero Init + Buffer), learning rates are differentially adjusted based on gradient sensitivity.

D.3 Core Training & Evaluation Commands

To facilitate reproduction, we provide the full environment dependencies, project structure, complete training scripts for Qwen2.5 and LLaMA-3.1, and batch evaluation scripts. All scripts are based on `train_gspo_buffer_multitask.py` and `evaluate_unified.py`.

D.3.1 Requirements & Structure

1. Installation Ensure specific versions of core libraries are installed:

```
pip install fastNLP==0.7.0
pip install torch==2.7.0
pip install transformers==4.51.0
```

2. Project Structure Ensure the relative positions of code and data are as follows:

```
.
|-- train_gspo_buffer_multitask.py    # Main training script
|-- evaluate_unified.py               # Main evaluation script
|-- unified_llm_model.py              # Model definition
|-- unified_utils.py                  # Utilities
|-- data_loader.py                    # Data loader
\-- data/                             # Data root
    |-- GSM8K/
    |-- ASDiv-Aug/
    |-- AQuA/
    |-- StrategyQA/
    \-- DU/
```

D.3.2 Training Commands

Adjust `batch_size` according to VRAM availability.

1. Qwen-2.5-7B-Instruct (Learning Rate: 1e-5)

(1) **GSM8K** (Budget: 1 Epoch)

```
python train_gspo_buffer_multitask.py \
    --model_id $QWEN \
    --data_path "./data/GSM8K" \
    --output_dir "./output_qwen_gsm8k" \
    --task_name "gsm8k" \
    --path_to_projection_module "None" \
    --num_thought_tokens 2 \
    --group_size 5 \
    --episodes_per_round 16 \
    --update_epochs 3 \
    --train_steps 150 \
    --save_every 10 \
    --lr 1e-5 \
    --max_data_epochs 1
```

(2) **ASDiv-Aug** (Budget: 1 Epoch)

```
python train_gspo_buffer_multitask.py \
    --model_id $QWEN \
    --data_path "./data/ASDiv-Aug" \
    --output_dir "./output_qwen_asdiv" \
    --task_name "asdiv-aug" \
    --path_to_projection_module "None" \
    --num_thought_tokens 2 \
    --group_size 5 \
    --episodes_per_round 16 \
    --update_epochs 3 \
    --train_steps 150 \
    --save_every 10 \
    --lr 1e-5 \
    --max_data_epochs 1
```

(3) **AQuA** (Budget: Unlimited / Fixed Steps)

```
python train_gspo_buffer_multitask.py \
    --model_id $QWEN \
    --data_path "./data/AQuA" \
    --output_dir "./output_qwen_aqua" \
    --task_name "aqua" \
    --path_to_projection_module "None" \
    --num_thought_tokens 2 \
    --group_size 5 \
    --episodes_per_round 16 \
    --update_epochs 3 \
    --train_steps 150 \
    --save_every 10 \
    --lr 1e-5 \
    --max_data_epochs 0
```

(4) **StrategyQA** (Budget: Unlimited / Fixed Steps)

```
python train_gspo_buffer_multitask.py \
    --model_id $QWEN \
    --data_path "./data/StrategyQA" \
    --output_dir "./output_qwen_sqa" \
    --task_name "strategyqa" \
    --path_to_projection_module "None" \
    --num_thought_tokens 2 \
    --group_size 5 \
    --episodes_per_round 16 \
    --update_epochs 3 \
    --train_steps 150 \
    --save_every 10 \
    --lr 1e-5 \
    --max_data_epochs 0
```

2. LLaMA-3.1-8B-Instruct (Learning Rate: 5e-6)

(1) **GSM8K** (Budget: 1 Epoch)

```
python train_gspo_buffer_multitask.py \
    --model_id $LLAMA \
    --data_path "./data/GSM8K" \
    --output_dir "./output_llama_gsm8k" \
    --task_name "gsm8k" \
    --path_to_projection_module "None" \
    --num_thought_tokens 2 \
    --group_size 5 \
    --episodes_per_round 16 \
    --update_epochs 3 \
    --train_steps 150 \
    --save_every 10 \
    --lr 5e-6 \
    --max_data_epochs 1
```

(2) **ASDiv-Aug** (Budget: 1 Epoch)

```
python train_gspo_buffer_multitask.py \
    --model_id $LLAMA \
    --data_path "./data/ASDiv-Aug" \
    --output_dir "./output_llama_asdiv" \
    --task_name "asdiv-aug" \
    --path_to_projection_module "None" \
    --num_thought_tokens 2 \
    --group_size 5 \
    --episodes_per_round 16 \
    --update_epochs 3 \
    --train_steps 150 \
    --save_every 10 \
    --lr 5e-6 \
    --max_data_epochs 1
```

(3) **AQuA** (Budget: Unlimited / Fixed Steps)

```
python train_gspo_buffer_multitask.py \
    --model_id $LLAMA \
    --data_path "./data/AQuA" \
    --output_dir "./output_llama_aqua" \
    --task_name "aqua" \
    --path_to_projection_module "None" \
    --num_thought_tokens 2 \
    --group_size 5 \
    --episodes_per_round 16 \
    --update_epochs 3 \
    --train_steps 150 \
    --save_every 10 \
    --lr 5e-6 \
    --max_data_epochs 0
```

(4) **StrategyQA** (Budget: Unlimited / Fixed Steps)

```
python train_gspo_buffer_multitask.py \
    --model_id $LLAMA \
    --data_path "./data/StrategyQA" \
    --output_dir "./output_llama_sqa" \
    --task_name "strategyqa" \
    --path_to_projection_module "None" \
    --num_thought_tokens 2 \
    --group_size 5 \
    --episodes_per_round 16 \
    --update_epochs 3 \
    --train_steps 150 \
    --save_every 10 \
    --lr 5e-6 \
    --max_data_epochs 0
```

D.3.3 Evaluation Commands

We use the following shell script template to batch evaluate 5 random seeds (41–45) and automatically log results.

```
#!/bin/bash

# ================ Configuration =================
MODEL_ID=$QWEN  # Or $LLAMA
TASK_NAME="gsm8k"
DATA_PATH="./data/GSM8K"
CKPT_PATH="./output_qwen_gsm8k/ckpt/final.bin"
LOG_DIR="./logs_eval_${TASK_NAME}"
mkdir -p $LOG_DIR
# ================================================

echo "Starting Evaluation for Task: $TASK_NAME"

for SEED in {41..45}; do
    echo "Running Seed: $SEED"
    python -u evaluate_unified.py \
        --model_id "$MODEL_ID" \
        --data_path "$DATA_PATH" \
        --params_file_name "$CKPT_PATH" \
        --task_name "$TASK_NAME" \
        --dataset_split "test" \
        --num_thought_tokens 2 \
        --seed $SEED \
        2>&1 | tee "${LOG_DIR}/eval_seed${SEED}.log"
done

echo "Evaluation finished."
```

Quick Result Extraction:

```
grep "Final Accuracy" ./logs_eval_gsm8k/*.log
```

E Additional Evaluation on Multi-Step Reasoning and Generalization

To validate the robustness of our Self-SoftCoT projection layer ($N = 2$) on more challenging multi-step reasoning tasks, we conducted evaluations on the MATH-500 (Lightman et al., 2023) and Gaokao2023 (Zhang et al., 2023) benchmarks. As shown in Table 9, even when applying features learned *exclusively* from elementary mathematics (GSM8K), Self-SoftCoT achieves a steady performance gain on MATH-500, improving zero-shot accuracy from 64.20% to 68.48%. These results suggest that the learned projection module transfers some reasoning-relevant signal beyond the training dataset, including to harder zero-shot benchmarks such as MATH-500.

Furthermore, to evaluate the practicality of real-world deployment, we analyze the Self-SoftCoT projection module from two perspectives: computational overhead and generalization capacity. First, the projection module is small enough for practical dynamic loading, with low additional module-loading overhead relative to the backbone.

Additionally, a single trained module exhibits cross-domain and difficulty generalization without necessitating a linear increase in computational overhead. We extracted the module trained exclusively on the GSM8K dataset and performed zero-shot transfer testing across distinct domains, including StrategyQA (Commonsense) and Date Understanding (Symbolic), alongside the aforementioned complex benchmarks. As Table 9 demonstrates, the GSM8K-trained module elevates StrategyQA zero-shot accuracy from 52.88% to 63.19%. This stable performance across disparate domains indicates that our projection layers learn transferable reasoning patterns rather than just memorizing domain-specific templates. Moreover, using the weights trained exclusively on GSM8K to initialize continual training on StrategyQA pushes accuracy to 69.08% while retaining an 88.42% accuracy on GSM8K. This demonstrates strong scalability and adaptability while effectively mitigating catastrophic forgetting.

F Comparison with Explicit Long-Chain Reasoning Models and Efficiency Analysis

Recently, explicit long-chain reasoning models (e.g., DeepSeek-R1 (DeepSeek-AI, 2025), QwenLongCoT (Qwen Team, 2025)) have demonstrated remarkable capabilities. To better understand the positioning of our proposed method, we conducted a benchmark comparison between the Native Qwen2.5-7B baseline, our Self-SoftCoT, and the reasoning-specialized DeepSeek-R1-Distill-Qwen-7B (DeepSeek-AI, 2025) under strictly identical single-GPU hardware and decoding settings.

We explicitly acknowledge that explicit long-chain models like DeepSeek-R1 exhibit stronger overall generalization and reasoning performance. As shown in Table 10, DeepSeek-R1 achieves significantly higher absolute accuracy on logically dense and complex tasks (such as AQuA and DU). However, the generation of thousands of explicit tokens inevitably introduces corresponding latency overheads (e.g., averaging roughly 46 seconds per query on AQuA).

In contrast, Self-SoftCoT is designed to explore a different trade-off: sacrificing explicit human readability in favor of inference efficiency. By relying on lightweight latent continuous vectors rather than generating long explicit reasoning traces, Self-SoftCoT introduces only modest additional latency compared with the vanilla baseline. Specifically, under identical hardware and decoding settings, our method shows lower end-to-end latency than DeepSeek-R1-Distill-Qwen-7B on StrategyQA (0.52s vs. 11.12s), AQuA (5.00s vs. 45.99s), DU (2.36s vs. 42.02s), and GSM8K (3.28s vs. 13.17s), with up to a $21\times$ speedup on StrategyQA. We emphasize that this efficiency gain mainly comes from avoiding the generation of long explicit reasoning traces and therefore using far fewer output tokens, rather than from any improvement in the model's per-token generation speed.

To further investigate whether our method yields diminishing returns on highly optimized baselines, we integrated Self-SoftCoT into the newly released Qwen3-8B architecture. As shown in Table 11, Self-SoftCoT consistently delivers improvements over the robust Native Qwen3-8B-Instruct (Qwen Team, 2025) baseline across multiple random seeds. We also list Qwen3-LongCoT (Qwen Team, 2025) as a reference. Unsurprisingly, Qwen3-LongCoT achieves a higher accuracy ceiling across these benchmarks. However, consistent with the DeepSeek-R1 analysis, its peak performance is fundamentally driven by generating massive amounts of explicit text trajectories, which consumes significantly more inference time.

We candidly acknowledge that, compared to state-of-the-art explicit reasoning models, Self-

Model Configuration / Training Pipeline	GSM8K (Math)	StrategyQA (Commonsense)	DU (Symbolic)	MATH-500 (Hard Math)	Gaokao2023 (Compr.)
Native Qwen2.5-7B-Instruct (Baseline, $N = 0$)	84.18% ± 1.60%	52.88% ± 5.07%	63.92% ± 5.80%	64.20% ± 1.22%	56.05% ± 2.86%
Zero-Shot Transfer (GSM8K Weights, $N = 2$)	88.89% ± 0.59%	63.19% ± 0.30%	67.68% ± 1.15%	68.48% ± 0.92%	58.49% ± 3.77%
Reference: Task-Specific Training	88.89% ± 0.59%	66.94% ± 1.30%	67.68% ± 1.15%	–	–
Continual Training (GSM8K Init. → StrategyQA)	88.42% ± 0.97%	69.08% ± 0.45%	69.36% ± 2.77%	–	–

Table 9: Cross-Domain & Difficulty Generalization. The base model is Qwen2.5-7B-Instruct. The Zero-Shot Transfer module ($N = 2$) was trained *solely* on GSM8K without any in-domain training on the target datasets. Since DU has no training split, its result is obtained by directly transferring the GSM8K-trained projection weights.

Dataset	Native Qwen2.5 (Baseline, $N = 0$)		Self-SoftCoT (Ours)		DeepSeek-R1-7B	
	Acc.	Avg. Tokens / Latency	Acc.	Avg. Tokens / Latency	Acc.	Avg. Tokens / Latency
GSM8K	84.00%	211 / 3.28s	89.39%	216 / 3.28s	89.69%	821 / 13.17s
ASDiv-Aug	88.63%	112 / 1.78s	90.37%	106 / 1.68s	91.81%	394 / 6.17s
AQuA	64.96%	267 / 4.49s	78.74%	320 / 5.00s	85.43%	2519 / 45.99s
StrategyQA	53.28%	54 / 0.86s	64.85%	32 / 0.52s	58.95%	701 / 11.12s
DU	66.40%	141 / 2.27s	68.40%	149 / 2.36s	86.40%	2361 / 42.02s

Table 10: Accuracy and inference efficiency benchmark among 7B-scale models. All results are evaluated on a single GPU under strictly identical hardware and decoding settings, using the same fixed random seed (seed = 41) for all compared methods.

SoftCoT exhibits inherent limitations. Because our approach strictly freezes the backbone model and updates a relatively small parameter footprint (only the projection layer for $N = 2$ soft tokens), its absolute reasoning capacity and broad cross-domain generalization naturally cannot rival explicit long-chain models that leverage large-scale optimization and massive dynamic token generation. On highly complex tasks that strictly require rigorous, multi-step symbolic deduction, explicit long-chain models still maintain a clear advantage.

Ultimately, by avoiding the need for resource-intensive full-parameter fine-tuning or large-scale data distillation, Self-SoftCoT does not seek to push the absolute upper bound of reasoning capability. Instead, it serves as a parameter-efficient alternative tailored for deployment scenarios with strict latency and computational constraints.

G Ablation Study on Reinforcement Learning Algorithms

G.1 Analysis of Policy Collapse with DPO and KTO

Applying preference optimization algorithms (e.g., DPO (Rafailov et al., 2023) or KTO (Ethayarajh et al., 2024)) to train the projection layer frequently leads to systematic policy collapse. This instability primarily arises from the lack of robust gradient clipping mechanisms. Without strict step-size constraints, unconstrained preference updates amplify minor gradient variances, causing rapid KL divergence from the reference policy and subsequent optimization failure.

G.2 Controlled Comparison: GSPO vs. GRPO

Although GRPO serves as a standard baseline for explicit text-based reasoning, it exhibits higher variance during projection layer updates compared to the sequence-level GSPO algorithm adopted in this work.

Mechanistic Differences The core distinction lies in the update granularity. GRPO utilizes token-level policy ratio updates, where sampling extreme-variance tokens can introduce gradient noise. In contrast, GSPO employs sequence-level ratio aggregation, evaluating the explicit reasoning chain as a complete unit with length normalization and strict clipping. This structural difference mitigates the training oscillations caused by token-level variance.

Model / Configuration	GSM8K	ASDiv-Aug	AQuA	StrategyQA	DU
Native Qwen3 (Baseline)	91.42% ± 0.64%	91.65% ± 0.28%	81.89% ± 1.08%	65.37% ± 0.55%	82.40% ± 1.70%
Self-SoftCoT (Ours)	92.35% ± 0.38%	92.52% ± 0.18%	83.07% ± 0.96%	70.74% ± 1.63%	84.64% ± 1.69%
Qwen3 LongCoT	95.94% ± 0.37%	93.56% ± 0.13%	87.96% ± 0.99%	74.63% ± 0.90%	92.56% ± 0.83%

Table 11: Performance validation on the Qwen3-8B architecture. All results are evaluated across 5 random seeds (from 41 to 45). Self-SoftCoT provides gains over the baseline, while Qwen3-LongCoT represents the performance ceiling achieved via explicit long-chain generation.

Empirical Validation and Variance Reduction Under an identical budget of 7,000 sampling steps, the model trained with GRPO achieves a final GSM8K accuracy of 88.52% ± 1.12%. While competitive, the GSPO mechanism further stabilizes the training process, achieving an accuracy of 88.89% ± 0.59% and effectively halving the performance standard deviation.

G.3 Advantage Normalization Edge Cases

To address scenarios where reward variance approaches zero (e.g., due to exceptionally simple or difficult prompts), we implement a dual-layer safety mechanism during GSPO optimization:

- **Zero-Variance Skip**: Mini-batches with near-zero reward variance are discarded, and back-propagation is deferred until sufficient discriminative samples are accumulated.
- **Multi-Epoch Buffer Update**: Filtered, high-variance valid data is retained in a buffer for reuse across consecutive epochs, thereby maximizing data utilization efficiency.

H Disentangling Performance Gains and Architectural Advantages

To directly address concerns regarding incremental contributions and clearly isolate the sources of our performance gains, we conducted comprehensive ablation studies separating the impact of the self-generated soft thoughts architecture from the GSPO optimization strategy (evaluated on Qwen2.5-7B, GSM8K). The results demonstrate that Self-SoftCoT represents a distinct approach in both latent reasoning design and optimization paradigm.

1. Dynamic Generation vs. Static Parameters As shown in Table 12, reverting to static parameters (Static P-Tuning (Liu et al., 2021)) drops accuracy to 80.45%, significantly below the vanilla baseline. This indicates that static templates fail to provide effective guidance and instead act as disruptive noise in multi-step reasoning. Our architecture's efficacy relies strictly on real-time contextual awareness and dynamic feature projection.

2. Why SFT Fails on Distinct Projections If we retain our dynamic architecture but apply standard SFT, performance drops to 76.45%. Our distinct projection layer demands strict train-test length consistency ($N = 2$), preventing the "length-mismatch" trick used in prior work. Furthermore, SFT's reliance on fixed ground-truth sequences rather than the model's self-generated latent states fails to optimize these position-aware parameters. GSPO overcomes this by elevating optimization to the sequence level, enabling effective adaptation of these weights.

3. Architectural Advantages Self-SoftCoT overcomes the limitations of the SoftCoT framework. It avoids reliance on highly verbose hard-text prompts and artificial length mismatches. By maintaining strict train-inference length consistency and allowing distinct update directions across token slots, Self-SoftCoT achieves superior accuracy (88.89%) with minimal placeholders. Detailed ablation results across various token counts and architectural configurations are provided in Table 5 of the main text.

H.1 Clarifying Inference Deployment Costs

Compared to previous dual-model architectures, Self-SoftCoT offers structural advantages for deployment:

- **VRAM Efficiency:** SoftCoT relies on an "Assistant-Reasoner" pipeline, requiring two separate models. Self-SoftCoT operates within a self-contained single model, reducing VRAM requirements for training and serving.
- **KV Cache Reuse Potential:** Compared with SoftCoT, our framework does not rely on an additional assistant prompt or verbose soft-token injection text. Instead, Self-SoftCoT inserts

Model / Configuration	Core Mechanism and Features	GSM8K Acc.
Native Baseline ($N = 0$)	No extra prompts, standard zero-shot	84.18% $\pm$ 1.60%
Static P-Tuning (Ablation)	Static, fixed learnable vectors	80.45% $\pm$ 1.22%
SoftCoT (Single-model)	Shared projection + SFT + Verbose prompt	85.84%
SoftCoT (Dual-model)	Shared projection + SFT + Verbose prompt	85.81% $\pm$ 1.82%
Self-SoftCoT (SFT only)	Dynamic features + Distinct projections + SFT	76.45% $\pm$ 1.33%
Self-SoftCoT (GRPO)	Dynamic features + Distinct projections + GRPO	88.52% $\pm$ 1.12%
Self-SoftCoT (Ours)	**Dynamic features + Distinct projections + GSPO**	**88.89% $\pm$ 0.59%**

Table 12: Optimization Paradigm and Core Mechanism Ablation.

only a small number of special soft-token placeholders at the end of the question. Because the main prompt prefix remains unchanged, this design theoretically enables direct reuse of the prefix KV cache when transitioning from the thinking phase to the final answer decoding, thereby offering the potential to reduce prefill latency.

I Self-SoftCoT Templates (Ours)

Appendix I: Self-SoftCoT Templates (Ours)

Description: We insert special placeholders to mark latent thought generation positions.
- **Qwen2.5:** <|box_start|>, <|box_end|>, <|endoftext|>
- **LLaMA-3.1:** Mapped to reserved special tokens to prevent conflicts.

1. Math Reasoning (GSM8K & ASDiv)

Solve the following math problem efficiently and clearly:
- For simple problems (2 steps or fewer): Provide a concise solution with minimal equation.
- For complex problems (3 steps or more): Use this step-by-step format:

Step 1: [Brief calculations] ## Step 2: [Brief calculations] ...

Regardless of the approach, always conclude with:

Therefore, the final answer is: \boxed{answer}. I hope it is correct.

Where [answer] is just the final number or expression that solves the problem.

Problem: {question} <|box_start|><|endoftext|><|endoftext|><|box_end|>

2. Commonsense (StrategyQA)

You are required to answer the following question with Yes or No: {question}
<|box_start|><|endoftext|><|endoftext|><|box_end|>

Therefore, the final answer is Yes or No?

3. Algebra MCQ (AQuA)

You are required to solve the following math multiple choices questions. In the multiple choices question, there are five options: A, B, C, D, and E, respectively. The correct answer that solves the problem is one of these options. Your job is to solve the problem and find the correct option. Here are the instructions for solving the problem efficiently and clearly:
- For simple problems (2 steps or fewer): Provide a concise solution with minimal equation.
- For complex problems (3 steps or more): Use this step-by-step format: ## Step 1 ...

Regardless of the approach, always conclude with the following sentence:

Therefore, the final answer is: \boxed{answer}. I hope it is correct.

Where [answer] is the option from A, B, C, D, and E.

Only one letter from A to E is accepted in the answer span.

Problem: {question} <|box_start|><|endoftext|><|endoftext|><|box_end|>

Options: (A) [Option Content] (B) [Option Content] (C) [Option Content] (D) [Option Content] (E) [Option Content]

4. Symbolic Reasoning (Date Understanding)

You are required to solve the following math multiple choices questions. In the multiple choices question, there are six options: A, B, C, D, E, and F, respectively. The correct answer that solves the problem is one of these options. Your job is to solve the problem and find the correct option. Here are the instructions for solving the problem efficiently and clearly:
- For simple problems (2 steps or fewer): Provide a concise solution with minimal description.
- For complex problems (3 steps or more): Use ## Step 1 ... format.

Regardless of the approach, always conclude with the following sentence:

Therefore, the final answer is: \boxed{answer}. I hope it is correct.

Where [answer] is the option from A, B, C, D, E, and F.

Only one letter from A to F is accepted in the answer span.

Problem: {question} <|box_start|><|endoftext|><|endoftext|><|box_end|>

Options: (A) ... (F) [Option Content]

Table 13: **Full Prompt Templates for Self-SoftCoT (Single-Stream Interface)**.

Appendix I.1: SoftCoT Baseline (Part 1: Assistant Model)

Description: Prompts used for the **Assistant Model** to generate soft tokens (hints).

1. Math Reasoning (GSM8K) - Assistant Input

You are required to generate 4 tokens to help another language model to solve the following math reasoning task efficiently and clearly. Here are the requirements of your generated tokens:

- The tokens should include some useful information for the reasoning problem, for example, the numbers and the operations needed for calculation.
- Generate the tokens starts from the most important or the highest related tokens.
- **Informative tokens are required**: (1) Do not need to generate a sentence or paragraph, (2) Do not need to generate the uninformative tokens such as serial number.
- The tokens should be useful for large language model to answer the question with the numbers.

Here is the problem: {question}.

2. Commonsense (StrategyQA) - Assistant Input

You are required to generate 4 tokens to help another language model to solve the following strategy reasoning task efficiently and clearly:

- The tokens should include some useful information for the reasoning problem.
- Generate the tokens starts from the most important or the highest related tokens.
- **Informative tokens are required**: (1) Do not need to generate a sentence or paragraph, (2) Do not need to generate the uninformative tokens such as serial number.
- The tokens should be useful for large language model to answer the question with `Yes` or `No`.

Here is the problem: {question}

3. Algebra MCQ (AQuA) - Assistant Input

You are required to generate 4 tokens to help another language model to solve the following math reasoning task efficiently and clearly. Here are the requirements of your generated tokens:

- The tokens should include some useful information for the reasoning problem, for example, the numbers and the operations needed for calculation.
- Generate the tokens starts from the most important or the highest related tokens.
- **Informative tokens are required**: (1) Do not need to generate a sentence or paragraph, (2) Do not need to generate the uninformative tokens such as serial number.
- The tokens should be useful for large language model to choose the correct option from A, B, C, D, and E.

Here is the problem: {question}.

4. Symbolic (Date) - Assistant Input

You are required to generate 4 tokens to help another language model to solve the following date understanding task efficiently and clearly.

- The tokens should include some useful information for the reasoning problem, for example, the numbers and the operations needed for calculation.
- Generate the tokens starts from the most important or the highest related tokens.
- **Informative tokens are required**: (1) Do not need to generate a sentence or paragraph, (2) Do not need to generate the uninformative tokens such as serial number.
- The tokens should be useful for large language model to answer the question with the correct option from A, B, C, D, E, and F.

Here is the problem: {question}.

Table 14: **SoftCoT Baseline Templates Part 1: Assistant Model Prompts**.

Appendix I.2: SoftCoT Baseline (Part 2: Main Model)

Description: The Main Model receives the soft tokens via a specific injection format.

1. Math Reasoning (GSM8K) - Main Input

Solve the following math problem efficiently and clearly:
- For simple problems (2 steps or fewer): Provide a concise solution with minimal equation.
- For complex problems (3 steps or more): Use this step-by-step format:

Step 1: [Brief calculations]
Step 2: [Brief calculations] ...
Regardless of the approach, always conclude with:
Therefore, the final answer is: \boxed{answer}. I hope it is correct.
Where [answer] is just the final number or expression that solves the problem.
Problem: {question}
There are some prompts generated by a weaker assistant model. Some prompts maybe useful while others maybe unuseful for your reasoning. If the prompts are correct, you can use it as reference. If the prompts are not correct, you can ignore them and focus back to solving the problem.
Here are prompts: `[SOFT_TOKENS]`

2. Commonsense (StrategyQA) - Main Input

You are required to answer the following question with `Yes` or `No`: {question}

Here is something useful for your reasoning: `[SOFT_TOKENS]`

Therefore, the final answer is `Yes` or `No`?

3. Algebra MCQ (AQuA) - Main Input

You are required to solve the following math multiple choices questions. In the multiple choices question, there are five options: A, B, C, D, and E, respectively. The correct answer that solves the problem is one of these options. Your job is to solve the problem and find the correct option. Here are the instructions for solving the problem efficiently and clearly:
- For simple problems (2 steps or fewer): Provide a concise solution with minimal equation.
- For complex problems (3 steps or more): Use this step-by-step format: ## Step 1 ...

Regardless of the approach, always conclude with the following sentence:
Therefore, the final answer is: \boxed{answer}. I hope it is correct.
Where [answer] is the option from A, B, C, D, and E.
Only one letter from A to E is accepted in the answer span.
Problem: {question}
There are some prompts generated by a weaker assistant model. Some prompts maybe useful while others maybe unuseful for your reasoning. If the prompts are correct, you can use it as reference. If the prompts are not correct, you can ignore them and focus back to solving the problem.
Here are prompts: `[SOFT_TOKENS]`

4. Symbolic (Date) - Main Input

You are required to solve the following math multiple choices questions. In the multiple choices question, there are six options: A, B, C, D, E, and F, respectively. The correct answer that solves the problem is one of these options. Your job is to solve the problem and find the correct option. Here are the instructions for solving the problem efficiently and clearly:
- For simple problems (2 steps or fewer): Provide a concise solution with minimal description.
- For complex problems (3 steps or more): Use this step-by-step format: ## Step 1 ...

Regardless of the approach, always conclude with the following sentence:
Therefore, the final answer is: \boxed{answer}. I hope it is correct.
Where [answer] is the option from A, B, C, D, E, and F.
Only one letter from A to F is accepted in the answer span.
Problem: {question}
There are some prompts generated by a weaker assistant model. Some prompts maybe useful while others maybe unuseful for your reasoning. If the prompts are correct, you can use it as reference. If the prompts are not correct, you can ignore them and focus back to solving the problem.
Here are prompts: `[SOFT_TOKENS]`

Table 15: **SoftCoT Baseline Templates Part 2: Main Model Prompts**.

J Input-Output Case Comparison

Appendix J: Input-Output Case Comparison

Description: Real-world comparison: **Zero-Shot CoT**, **SoftCoT**, and **Self-SoftCoT (Ours)**.

Case 1: Cecilia's Puppy (Multi-stage Logic)

Problem: Cecilia just bought a new puppy. According to her veterinarian, she has to feed the puppy 1 cup of dog food every day for the first 180 days. Then she has to feed the puppy 2 cups of dog food every day for the rest of its life. If one bag of dog food contains 110 cups, how many bags of dog food will Cecilia use in the first year?

Baseline: Zero-Shot CoT ($N = 0$) **Incorrect**

Step 1: Calculate the amount of dog food needed for the first year.
- In the first year, Cecilia has to feed the puppy 1 cup... for 365 days.
- Total... $1 \times 365 = 365$ cups. *(Error: Ignores the diet change after 180 days)*

Therefore, the final answer is: $\boxed{365}$.

Ref: SoftCoT ($N = 4$) **Incorrect**

Step 1: ... for the first 180 days... $180 \times 1 = 180$ cups.
Step 2: Determine if any dog food is needed beyond...
- ... 185 days remain... but they are not needed until the second year, we do not count them in the first year's total. *(Error: Logic interruption)*

Therefore, the final answer is: $\boxed{180}$.

Ours: Self-SoftCoT ($N = 2$) **Correct**

Step 1: Calculate... for the first 180 days... Total is 180 cups.
Step 2: Calculate the remaining days... $365 - 180 = 185$ days.
Step 3: Calculate... for the remaining days... $2 \times 185 = 370$ cups.
Step 4: Calculate the total... $180 + 370 = 550$ cups.
Step 5: Determine how many bags... $550/110 = 5$ bags.

Therefore, the final answer is: $\boxed{5}$.

Why Self-SoftCoT Succeeds:

Self-SoftCoT correctly decomposes the timeline into two distinct stages (first 180 days vs. remaining 185 days) and accurately integrates the implicit constraint (1 year = 365 days), maintaining the logical chain that both baselines failed to uphold.

Case 2: Brady's Piggy Bank (Arithmetic Precision)

Problem: Brady is counting the money in his piggy bank. He has 100 pennies, 40 nickels, 20 dimes, and 40 pieces of dollar bills. How much does Brady have in his piggy bank in dollars?

Baseline: Zero-Shot CoT ($N = 0$) **Incorrect**

... Total is 46 dollars. *(Error: Arithmetic hallucination,* $1 + 2 + 2 + 40 = 45$*)*

Ref: SoftCoT ($N = 4$) **Incorrect**

... Total is 43 dollars. *(Error: Calculation error, sum is 45)*

Ours: Self-SoftCoT ($N = 2$) **Correct**

... Step 5: Add up all the values... $1 + 2 + 2 + 40 = 45$ dollars.

Therefore, the final answer is: $\boxed{45}$.

Why Self-SoftCoT Succeeds:

Self-SoftCoT demonstrates superior calculation stability. While both baselines correctly identified the individual values, they failed at the final aggregation step due to arithmetic hallucinations. Self-SoftCoT maintained precise state tracking for the summation.

Case 3: John's Driving Cost (Commonsense & Hallucination)

Problem: John hires a driving service to get him to work each day. His work is 30 miles away and he has to go there and back each day. He goes to work 5 days a week for 50 weeks a year. He gets charged $2 per mile driven and he also gives his driver a $150 bonus per month. How much does he pay a year for driving?

Baseline: Zero-Shot CoT ($N = 0$) **Incorrect**

... round trip is 60 miles. ... total daily mileage is $60 \times 2 = 120$ miles. *(Error: Double counting)*

... total annual cost... $60000 + 1800 = 61800$. Answer: $\boxed{61800}$.

Ref: SoftCoT ($N = 4$) **Incorrect**

... for 50 weeks, which means **10 months**... Annual bonus is $150 \times 10 = 1500$. *(Error: Commonsense failure, 1 year = 12 months)*

Therefore, the final answer is: $\boxed{31500}$.

Ours: Self-SoftCoT ($N = 2$) **Correct**

Step 1: Total miles... $60 \times 250 = 15000$ miles. Cost... 30000.
Step 2: ... there are 12 months in a year, his annual bonus is $150 \times 12 = 1800$ dollars.

Step 3: ... $30000 + 1800 = 31800$. Answer: $\boxed{31800}$.

Why Self-SoftCoT Succeeds:

Self-SoftCoT correctly interprets the semantic structure ("there and back") avoiding double-counting, and successfully grounds commonsense knowledge (1 year = 12 months) despite the misleading context ("50 weeks"), which trapped SoftCoT.

Table 16: **Input-Output Case Comparison on GSM8K (Full Details)**.