C-Thinker: Latent Protocol Co-evolution for Dual-Agent Reasoning

Anonymous ACL submission

Abstract

Implicit reasoning and agent self-evolution offer promising research directions for enhancing the complex logic capabilities of Large Language Models. However, existing latent-space reasoning is often constrained by unidirectional communication, and agent self-evolution frequently suffers from delayed updates and restricted interactions due to a lack of real-time state feedback. To address these challenges, we propose **C-Thinker**, a dual-agent reasoning framework based on **Latent Protocol** Co-evolution. This framework reconstructs reasoning as a dynamic closed-loop interaction between a frozen Primary Model ($\mathcal{M}_R$) and an Auxiliary Model ($\mathcal{M}_T$). Specifically, the $\mathcal{M}_T$ leverages real-time feedback from the $\mathcal{M}_R$'s underlying hidden states to generate a Latent Protocol, which is then injected into the $\mathcal{M}_R$ to guide subsequent reasoning steps. C-Thinker employs a randomly initialized, lightweight $\mathcal{M}_T$ to minimize pre-trained semantic constraints, while guiding the stable convergence of the Latent Protocol through Semantic Consistency Regularization and Intrinsic Evolution objectives. Experiments demonstrate that C-Thinker achieves SOTA or highly competitive results across multiple reasoning benchmarks. On the Tibetan-CoT dataset constructed from Ti-MMLU (Gao et al., 2025a), C-Thinker variants show substantial potential in low-resource cross-lingual reasoning, although the full bidirectional variant is affected by noisy low-resource feedback and does not outperform the unidirectional variant in this setting.

1 Introduction

Large Language Models (LLMs) (Yang et al., 2025; DeepSeek-AI et al., 2025; Team et al., 2025a; OpenAI, 2023; Dubey et al., 2024; Zeng et al., 2024; Team et al., 2025b) have demonstrated strong capabilities across tasks such as mathematics, symbolic logic, and commonsense reasoning. However, their reasoning stability remains insufficient in scenarios requiring multi-step derivation, intermediate state correction, and cross-lingual transfer. To further enhance complex reasoning capabilities, existing research has proposed various approaches, including **Test-time Scaling (TTS)**(Zhang et al., 2025b), reinforcement learning(DeepSeek-AI et al., 2025), multi-agent self-evolution(Gao et al., 2025b), and implicit reasoning(Hao et al., 2024). Among these, implicit reasoning enables modeling intermediate processes in a continuous space, while multi-agent self-evolution emphasizes improving reasoning through interaction and feedback.

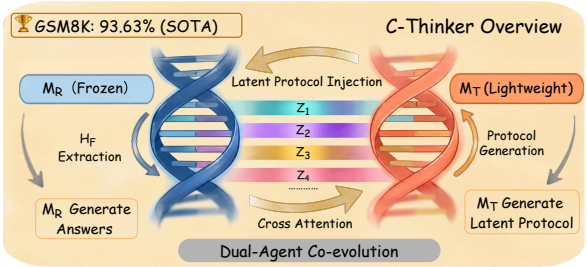


Figure 1: Overall overview of C-Thinker.

However, existing methods(Cao et al., 2024; Goyal et al., 2019; Deng et al., 2022; Casper et al., 2023) still face significant limitations. Self-evolution methods mediated by discrete text often suffer from coarse feedback granularity and delayed update processes, such as TextGrad(Yuksekgonul et al., 2024), AdaSTaR(Koh et al., 2025), and SPC(Chen et al., 2025). While latent-space reasoning methods bypass discrete text constraints, such as Coconut (Hao et al., 2024) and SoftCoT(Xu et al., 2025), they predominantly employ unidirectional static injection. Consequently, latent representations struggle to adapt in real-time to the current state of $\mathcal{M}_R$, making them prone to semantic drift and logical misalignment. Thus, a mechanism for continuous correction of intermediate states in complex reasoning remains absent. Although LatentEvolve attempts

to combine self-evolution with latent-space reasoning, it does not fully decouple from discrete text constraints and still struggles with update latency.

To mitigate semantic drift and update latency, we propose **C-Thinker**, which combines the strengths of implicit reasoning and multi-agent self-evolution. An overview of C-Thinker is shown in Figure 1. This approach reconstructs implicit reasoning as a dynamic interaction between a $\mathcal{M}_R$ and an $\mathcal{M}_T$, utilizing hidden state feedback from $\mathcal{M}_R$ to guide $\mathcal{M}_T$ in generating continuously adjustable latent representations. Following the observation in SoftCoT++ that high variance helps escape pre-trained semantic priors, expand the search neighborhood, and maintain diversity, C-Thinker employs a frozen $\mathcal{M}_R$ and a randomly initialized, lightweight $\mathcal{M}_T$. Furthermore, we introduce Semantic Consistency Regularization and Intrinsic Evolution objectives to enhance stability of latent representations. Experimental results show this method provides consistent gains in complex reasoning and low-resource cross-lingual scenarios.

Our main contributions are as follows:

1. We propose a bidirectional interactive reasoning architecture based on state feedback. By introducing multi-agent self-evolution into the latent space and training a dynamic Latent Protocol, this approach addresses semantic drift, logical misalignment, and update latency.

2. We design a joint learning objective based on a self-evolution mechanism. Through Semantic Consistency Regularization and Intrinsic Evolution via feature selection, the framework ensures the effective convergence of the Latent Protocol within the continuous space.

3. C-Thinker achieves competitive results across multiple complex benchmarks and demonstrates promising cross-lingual logic transfer on **Tibetan-CoT**, a low-resource cross-lingual dataset constructed in this study.

2 Related Work

Latent Space Reasoning Latent Space Reasoning aims to achieve efficient logical reasoning within continuous vector spaces. **Coconut**(Hao et al., 2024), **CODI**(Shen et al., 2025), and **CCOT**(Cheng and Van Durme, 2024) investigate diverse implicit reasoning paradigms by evolving implicit hidden states, aligning explicit and implicit representations via self-distillation, and compressing Chains-of-Thought (CoT) through variable-length continuous deliberation tokens. **SoftCoT**(Xu et al., 2025) introduces a lightweight injection framework that guides a frozen $\mathcal{M}_R$ using Soft Thoughts. Furthermore, **LatentEvolve**(Zhang et al., 2025a), inspired by the Complementary Learning Systems (CLS) theory of the human brain, facilitates continuous Latent Space evolution during test-time via daytime fast adaptation and night-time offline consolidation. Despite these advancements, most existing works rely on unidirectional injection or asynchronous updates, which struggle with Semantic Drift and the challenge of real-time feedback alignment between dual agents during dynamic reasoning interaction.

Self-evolving Agents Self-evolving agents has achieved iterative capability enhancement through interaction, following three primary evolutionary trajectories. The first category comprises In-Context Learning (ICL)-based evolution, such as **TextGrad**(Yuksekgonul et al., 2024) and **AgentSquare**(Shang et al., 2024), which utilize textual gradient feedback for component optimization or automate the search for optimal architectures within modular spaces, enabling policy iteration without parameter updates. The second category involves Supervised Fine-Tuning (SFT)-based evolution, such as **SiriuS**(Zhao et al., 2025) and **AdaSTaR**(Koh et al., 2025), which facilitate self-improvement through guided reasoning or optimize self-taught reasoning via adaptive data sampling, thereby forming a closed-loop capability evolution. The third category encompasses Reinforcement Learning (RL)-based evolutionary paradigms, including **ReMA**(Wan et al., 2025), **GiGPO**(Feng et al., 2025), and **HGPO**(He et al., 2026), which introduce multi-agent meta-reasoning, Group Relative Policy Optimization, or Hierarchical Group Policy Optimization to drive autonomous alignment and capability evolution in complex decision spaces or long-horizon tasks. **C-Thinker** extends these evolutionary principles to latent space interactions, facilitating deeper logical alignment via the Co-evolution of the Latent Protocol.

3 Methodology

3.1 Problem Formulation

Given the limitations of existing latent-space reasoning in unidirectional generation and dynamic interaction, C-Thinker reconstructs implicit reasoning as a **Co-evolution** process between Primary and

Auxiliary agents under a **Latent Protocol** framework. Traditional methods typically treat implicit generation as a static mapping $g(x) \mapsto \mathbf{z}$ that is independent of $\mathcal{M}_R$'s internal states. This isolated generation approach makes it difficult to establish robust communication standards between the Primary and Auxiliary Models. In our framework, given an input sequence x and target output y, we define the conditioned latent vector generation function as:

$$\mathbf{z} = g_\phi\left(x, \mathcal{F}_{\mathcal{M}_R}(x)\right), \quad \mathbf{z} \in \mathbb{R}^{L \times d_R} \tag{1}$$

where $\mathcal{F}_{\mathcal{M}_R}(x)$ denotes the underlying hidden states extracted by $\mathcal{M}_R$ after encoding the input x, and $\mathbf{z}$ represents the Latent Protocol features generated by $\mathcal{M}_T$. Subsequently, $\mathcal{M}_R$ performs autoregressive decoding based on the augmented input containing the protocol features: $y \sim \mathcal{M}_R(x, \mathbf{z})$.

In the specific architecture, $\mathcal{M}_R$ remains parameter-frozen, while $\mathcal{M}_T$ utilizes a randomly initialized, lightweight Transformer Block. During the training process, only $\mathcal{M}_T$ and the associated projection parameters are updated to generate the Latent Protocol features used for decoding by $\mathcal{M}_R$. C-Thinker framework is illustrated in Figure 2.

3.2 Bidirectional Interactive Architecture

The inference architecture of C-Thinker comprises three modules: State Feedback Extraction, Interactive Protocol Generation, and Representation Injection.

Representational Feedback Extraction. For an input sequence x, we first pass it through the $\mathcal{M}_R$ and extract the hidden states from its initial Transformer layers as the feedback signal $\mathbf{H}_F \in \mathbb{R}^{T \times d_R}$:

$$\mathbf{H}_F = \text{Layer}_1(x) \tag{2}$$

We select the hidden states of the first layer as the feedback signal, as this layer preserves substantial lexical-level information and input constraints, making it suitable as a conditional input for subsequent protocol generation.

Interactive Protocol Generation. The core objective of the $\mathcal{M}_T$ is to integrate its own reasoning intent with the feedback signal from $\mathcal{M}_R$. At each layer, $\mathcal{M}_T$ first constructs an initial reasoning representation through self-attention, and then introduces a Cross-Attention Layer for state interaction. The process is defined as follows:

$$\widetilde{\mathbf{H}}_T = \text{S-A}(\text{RMSNorm}(\mathbf{H}_T^{(l-1)})) + \mathbf{H}_T^{(l-1)} \tag{3}$$

$$\mathbf{H}_T^{(l)} = \text{C-A}(Q = \widetilde{\mathbf{H}}_T, K = \mathbf{H}_F, V = \mathbf{H}_F) + \widetilde{\mathbf{H}}_T \tag{4}$$

The query vector Q is derived from the current hidden states of $\mathcal{M}_T$. The key-value pairs (K, V) are provided by the feedback signal $\mathbf{H}_F$ from $\mathcal{M}_R$. In each layer, $\mathcal{M}_T$ fuses this feedback to update the protocol representation.

Representational Injection. The final state $\mathbf{H}_T$ produced after multi-layer interaction contains aligned reasoning information. We use a linear projection layer $\mathcal{P}_\theta$ to map it to the hidden dimension of the $\mathcal{M}_R$, yielding the final protocol vector sequence $\mathbf{Z}$:

$$\mathbf{Z} = \mathcal{P}_\theta(\mathbf{H}_T), \quad \mathbf{Z} \in \mathbb{R}^{L \times d_R} \tag{5}$$

Subsequently, we place $\mathbf{Z}$ into the reserved latent slots $\mathcal{I}_z$ of the $\mathcal{M}_R$ and use it as an additional continuous representation in the subsequent autoregressive decoding process.

3.3 Self-Evolutionary Learning Objectives

We adopt a joint training framework consisting of a base supervised objective, a Semantic Consistency Regularization term, and an Intrinsic Evolution objective. The overall optimization objective is defined as:

$$\mathcal{L}_{total} = \lambda_{sft}\mathcal{L}_{SFT} + \lambda_{align}\mathcal{L}_{align} + \lambda_{evo}\mathcal{L}_{evo} \tag{6}$$

where λ_{sft}, λ_{align}, and λ_{evo} are hyperparameters that balance the different loss terms.

Base Supervision Objective ($\mathcal{L}_{SFT}$). This term ensures that $\mathcal{M}_R$ can stably decode the correct reasoning trajectory and final answer after receiving the interactively generated protocol features $\mathbf{z}$. We optimize this using the standard autoregressive negative log-likelihood (NLL) loss:

$$\mathcal{L}_{SFT} = -\sum_t m_t \log p_{\mathcal{M}_R}(y_t \mid x, \mathbf{z}, y_{<t}) \tag{7}$$

where m_t is a masking operator that ensures the loss is computed only over tokens in the target output.

Semantic Consistency Regularization (L_{align}). In a continuous space without constraints from a discrete vocabulary, unconstrained protocol updates can easily drive features away from semantically valid regions. To suppress such semantic drift, we introduce a KL-divergence-based consistency

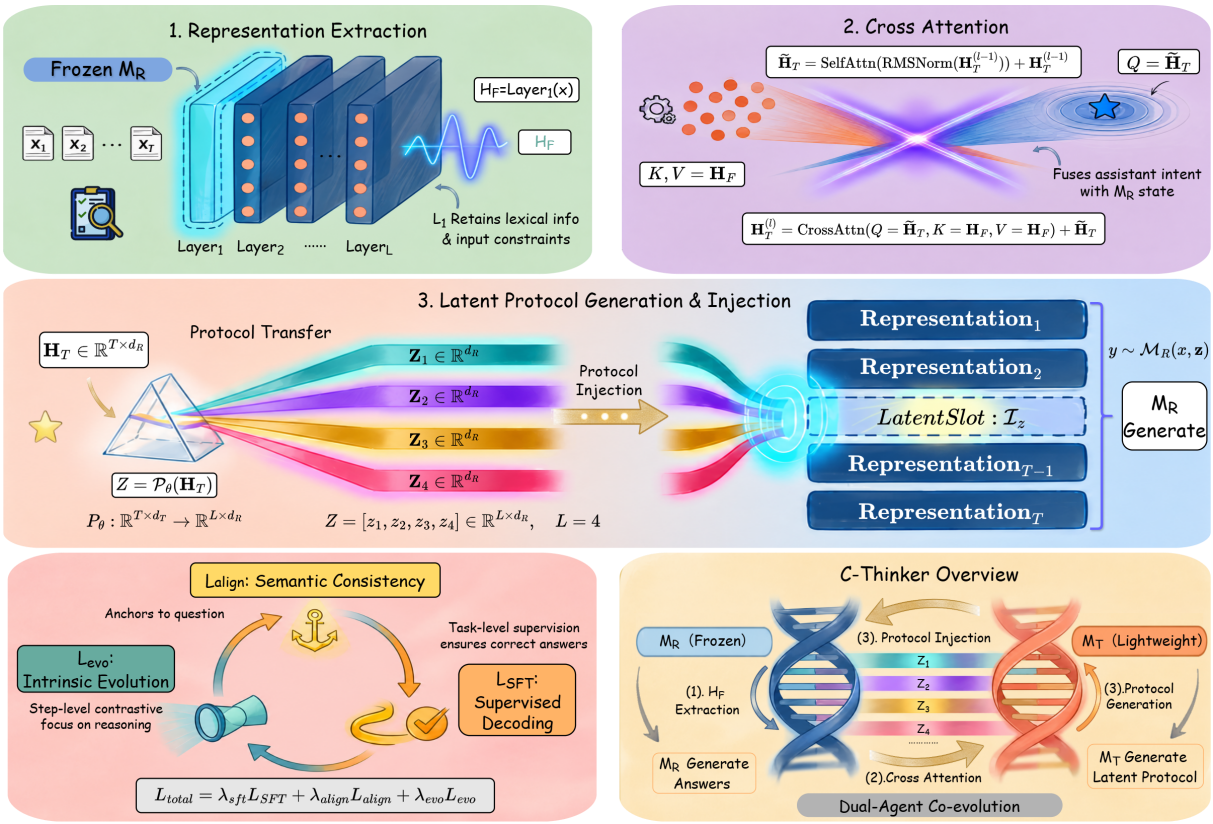


Figure 2: The C-Thinker framework. First, hidden states $\mathbf{H}_F$ are extracted from the initial layers of the frozen $\mathcal{M}_R$ as state feedback. Subsequently, $\mathcal{M}_T$ fuses these states through self-attention and cross-attention to generate Latent Protocol representations. Finally, the projected protocol vectors $\mathbf{Z}$ are injected into the latent slots of the Primary Model for decoding. During the training phase, the Latent Protocol is jointly optimized using supervised decoding objectives, semantic consistency regularization, and intrinsic evolution goals.

constraint that anchors the generated Latent Protocol to the intent of the question. Let $\mathbf{h}_q$ denote the projected representation of the question intent in the input sequence, and let $\mathbf{z}_j$ denote the generated representation of the j-th protocol token:

$$\mathcal{L}_{align} = \frac{1}{L} \sum_{j=1}^{L} D_{KL}\left(\mathrm{softmax}(\mathbf{h}_q) \| \mathrm{softmax}(\mathbf{z}_j)\right) \tag{8}$$

This term contracts the feasible search region of protocol updates during training, ensuring that the $\mathcal{M}_T$ explores consistently around the semantics of the current question.

Intrinsic Evolution Objective (L_{evo}). This term uses Teacher CoT to construct step-level contrastive supervision, guiding $\mathcal{M}_T$ to focus on reasoning steps contributing more to the final answer.

Specifically, we first encode the teacher reasoning chain into a set of step representations $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_K$, then match each step against the gold answer representation $\mathbf{a}$, and select the step with the highest similarity to the answer as the positive anchor: $k^\star = \arg\max_k \cos(\mathbf{s}_k, \mathbf{a})$;

We then construct a step distribution based on the similarity between the question-intent representation $\mathbf{h}_q$ and each teacher step representation $\mathbf{s}_k$, namely $\pi_\phi(k \mid x) = \mathrm{softmax}\left(\frac{\cos(\mathbf{h}_q, \mathbf{s}_k)}{\tau}\right)$, and increase the probability assigned to the positive anchor $k^\star$ by minimizing the following objective:

$$\mathcal{L}_{evo} = -\log \pi_\phi(k^\star \mid x) \tag{9}$$

These three losses play complementary roles. L_{SFT} provides task-level supervision to ensure that the protocol features support the final decoding process. L_{align} constrains the protocol representation to remain aligned with the question semantics, thereby reducing ineffective search regions in continuous space. Within this constrained space, L_{evo} performs step-level feature selection, making protocol vectors concentrate on reasoning states relevant to the answer. As a result, these Latent Protocol features become coupled with $\mathcal{M}_R$ decoding, enabling co-evolution of the two agents.

4 Experiments

To validate C-Thinker on complex reasoning tasks and the advantages of its bidirectional interactive architecture, we conduct extensive evaluations on multiple benchmark datasets. This section presents the experimental setup, results, ablation studies, and analysis of latent-space training dynamics.

4.1 Experimental Setup

Datasets We evaluate C-Thinker across three dimensions: mathematical reasoning, commonsense and symbolic reasoning, and low-resource cross-lingual reasoning. Mathematical reasoning benchmarks include GSM8K(Cobbe et al., 2021), GSM8K-zh(Yu et al., 2023), AQuA(Ling et al., 2017), MATH-500(Hendrycks et al., 2021), AIME24(Zhang and Math-AI, 2024), and AIME25(Zhang and Math-AI, 2025). Commonsense and symbolic reasoning benchmarks include StrategyQA(Geva et al., 2021) and Date Understanding (DU) from BIG-bench(Srivastava et al., 2023). Furthermore, we introduce **Tibetan-CoT**, a Tibetan general multiple-choice Q&A dataset. This dataset features low-resource language characteristics and high cross-lingual logical complexity. Since Tibetan representations are sparse in mainstream pre-training corpora, this benchmark tests whether C-Thinker can bypass surface-level linguistic differences and achieve stable cross-lingual logical alignment via the Latent Protocol under extreme insufficient-representation conditions.

Tibetan-CoT is constructed based on Tibetan multiple-choice questions from the Ti-MMLU_subset670(Gao et al., 2025a). First, we cleaned and filtered the original questions, retaining only samples with complete stems, stably parsable A-D options, and valid ground-truth answers, resulting in 668 base questions. Subsequently, we used GPT-5.4 to generate five distinct styles of Tibetan reasoning processes for each question, requiring 3–5 lines of rationale ending with a fixed answer sentence. Ultimately, 550 original questions were successfully augmented, forming 2,750 samples. During data partitioning, we split samples by original question groups rather than augmented items; 110 original questions were reserved as a held-out evaluation set, while 2,200 augmented samples from the remaining 440 questions were used for training. This prevents augmented versions of the same question from appearing in both training and evaluation sets.

Baselines All experiments use Qwen2.5-7B and Qwen3-8B as $\mathcal{M}_R$. The compared methods cover three representative paradigms. The first consists of conventional discrete-text reasoning methods, including Base LLM, Zero-Shot CoT, Self-Consistency, and Self-Refine. The second consists of reinforcement learning methods, including GRPO, Reinforce(Zhang et al., 2021), and Reinforce++(Hu, 2025). The third consists of continuous latent-space reasoning methods, including Coconut, SoftCoT, SoftCoT-ZSAC, SoftCoT++ZSAC, SoftCoT++, and LatentEvolve. In addition, we include **C-Thinker UB** (Unidirectional Baseline) as a degraded baseline by removing the $\mathcal{M}_R$ state feedback path ($\mathbf{H}_F$) and the cross-attention mechanism, while retaining only one-way protocol generation, to analyze the actual contribution of bidirectional interaction to Latent Protocol learning.

Hyperparameters The $\mathcal{M}_T$ is implemented as a randomly initialized lightweight Transformer block. During training, the batch size is set to 2, and the loss weights in the joint optimization objective are set to $\lambda_{sft} = 0.33$, $\lambda_{align} = 0.33$, and $\lambda_{evo} = 0.33$. The warmup ratio is set to 0.1. All models are trained for joint representation learning on 8 NVIDIA A100 40G GPUs.

4.2 Main Results

Tables 1 and 2 present the results of C-Thinker on general reasoning tasks and complex mathematical reasoning tasks, respectively. Overall, C-Thinker demonstrates a consistent advantage across both Qwen models. Under the high-sampling setting of N=10, it outperforms all existing public baselines listed in the table. Taking Qwen3-8B as an example, C-Thinker reaches 94.31%, 87.80%, 71.40%, and 89.97% on GSM8K, AQuA, StrategyQA, and DU, respectively, at N=10, all exceeding methods such as SoftCoT++. Specifically, it improves upon SoftCoT++ by 0.66, 3.71, and 1.81 percentage points on GSM8K, AQuA, and DU, while also slightly outperforming its 71.22% on StrategyQA. However, StrategyQA is an exception to the general sampling-gain trend: C-Thinker obtains 73.14% at N=1 but 71.40% at N=10, suggesting that majority voting may amplify inconsistent sampled answers on this dataset. Furthermore, in the single-pass setting, C-Thinker achieves 87.01% and 73.14% on AQuA and StrategyQA, surpassing the N=10 results of SoftCoT++ (84.09% and 71.22%). Its 93.63% on GSM8K is virtually on par with the

Base LLM	Method	GSM8K	AQuA	StrategyQA	DU
Qwen2.5-7B	Base LLM	85.40	-	-	-
	ZS CoT-SC ($N = 1$)	83.70	64.53	49.65	66.40
	SoftCoT-ZSAC ($N = 1$)	84.85	64.96	52.71	67.04
	SoftCoT-SC ($N = 1$)	85.81	72.44	60.61	67.52
	C-Thinker UB	87.86	75.98	67.47	67.75
	C-Thinker ($N = 1$)	**88.70**	**77.56**	**69.43**	**79.13**
	C-Thinker ($N = 10$)	**93.03**	**82.68**	**80.35**	**83.20**
Qwen3-8B	Base LLM	89.84	-	-	-
	ZS CoT-SC ($N = 10$)	92.22	76.77	70.96	84.56
	SoftCoT++ZSAC ($N = 10$)	92.68	76.77	70.92	84.80
	Coconut-SC ($N = 10$)	90.37	76.38	-	-
	SoftCoT-SC ($N = 10$)	93.19	80.63	71.18	87.20
	SoftCoT++ ($N = 10$)	93.65	84.09	71.22	88.16
	C-Thinker UB ($N = 1$)	93.25	85.04	71.83	84.55
	C-Thinker ($N = 1$)	93.63	**87.01**	**73.14**	86.72
	C-Thinker ($N = 10$)	**94.31**	**87.80**	71.40	**89.97**

Table 1: Accuracy (%) comparison between C-Thinker and baseline methods on reasoning datasets.

latter's 93.65%. This indicates that performance gains do not primarily rely on multi-path voting but stem from more effective Latent Protocol interactions driven by bidirectional state feedback. Similarly, on Qwen2.5-7B, C-Thinker improves results at N=1 to 88.70%, 77.56%, 69.43%, and 79.13% for GSM8K, AQuA, StrategyQA, and DU, representing increases of 0.84%, 1.58%, 1.96%, and 11.38 percentage points over the unidirectional version. When the sample count increases to N=10, results reach 93.03%, 82.68%, 80.35%, and 83.20%, likewise outperforming baselines and demonstrating cross-model stability.

Table 2 further reinforces this trend through mathematical reasoning benchmarks. On **Qwen3-8B**, C-Thinker achieves 93.63% on GSM8K, 82.60% on MATH-500, 33.33% on AIME24, and 30.00% on AIME25 under the N=1 setting, all of which surpass the Base LLM and LatentEvolve in the same table. When the sampling count increases to N=8, performance further improves to 94.31%, 85.20%, 36.67%, and 33.33%, respectively. On **Qwen2.5-7B**, C-Thinker's N=1 results do not always hold an advantage over RL or TTT baselines, particularly on MATH-500, where it falls behind GRPO, TTRL, and LatentEvolve. However, under the N=8 configuration, C-Thinker reaches 93.03% on GSM8K and 78.20% on MATH-500, while achieving 23.33% and 13.33% on AIME24 and AIME25, matching the best or tied-best results. Overall, Table 2 suggests that while C-Thinker provides more stable gains on Qwen3-8B, its performance on Qwen2.5-7B is not always superior to reinforcement learning or test-time training baselines when N=1. Nevertheless, it achieves SOTA results across complex mathematical reasoning datasets as the sampling budget increases. This may be due to the weaker reasoning capabilities of Qwen2.5-7B, where the Latent Protocol struggles to fully compensate for instability in a single decoding pass at N=1. In contrast, at N=8, multi-path sampling allows more thorough utilization of candidate answer improvements provided by the Latent Protocol.

4.3 Ablation Study

This section analyzes the performance sources of C-Thinker through component ablation and cross-lingual supplementary experiments. Table 3 investigates the impact of the state feedback path, joint learning objectives, Auxiliary Model architecture, and sampling strategies on GSM8K. Table 4 further compares performance of different variants on Tibetan-CoT, GSM8K-ZH, and GSM8K to examine alignment stability in cross-lingual settings, particularly in low-resource scenarios.

Table 3 shows that the state feedback path and the complete joint objectives constitute the key sources of performance improvement. On GSM8K, the full model achieves 93.63% in the single-pass setting, representing a 0.38 percentage point increase over C-Thinker UB (93.25%) and a 1.15 percentage point improvement over SoftCoT-SC

Base LLM	Method	GSM8K	MATH-500	AIME24	AIME25
Qwen2.5-7B	Base LLM	85.40	55.80	0.00	0.00
	GRPO	92.30	75.85	6.67	3.33
	Reinforce++	92.60	75.02	13.33	6.67
	SoftCoT	80.13	65.80	3.33	0.00
	LatentSeek	91.58	66.20	10.00	3.33
	TTRL	92.80	77.39	**23.33**	**13.33**
	LatentEvolve	92.98	77.60	**23.33**	10.00
	C-Thinker $(N = 1)$	88.70	67.40	13.33	10.00
	C-Thinker $(N = 8)$	**93.03**	**78.20**	**23.33**	**13.33**
Qwen3-8B	Base LLM	89.84	55.20	3.33	3.33
	LatentEvolve	90.45	73.80	26.67	23.33
	C-Thinker $(N = 1)$	**93.63**	**82.60**	**33.33**	**30.00**
	C-Thinker $(N = 8)$	**94.24**	**85.20**	**36.67**	**33.33**

Table 2: Accuracy (%) comparison between C-Thinker and baseline methods on challenging mathematical reasoning benchmarks. In LatentEvolve, the number of samples used for Monte Carlo approximation of the policy gradient during the test-time latent space optimization process is 8.

Variant	Accuracy (%)
SFT-only	92.95
w/o L_{evo}	92.87
w/o L_{align}	92.65
Linear-Assistant	92.95
SoftCoT-SFT + Con + KL	92.27
SoftCoT-SC $(N = 1)$	92.48
SoftCoT++ $(N = 1)$	92.48
SoftCoT-SC $(N = 10)$	93.19
SoftCoT++ $(N = 10)$	93.65
SoftCoT++ $(N = 100)$	94.12
C-Thinker UB $(N = 1)$	93.25
C-Thinker UB $(N = 10)$	94.24
C-Thinker (Full, $N = 1$**)**	93.63
C-Thinker (Full, $N = 10$**)**	94.31

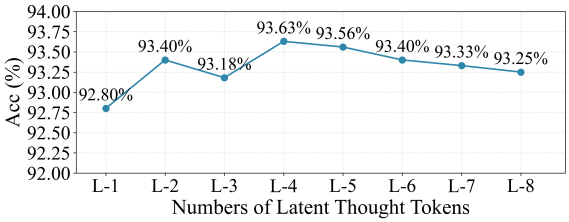
Table 3: Ablation results on GSM8K (Qwen3-8B).

and SoftCoT++ (92.48%) under the same sampling budget. When the sample count increases to N=10, the full model further improves to 94.31%, exceeding the 94.24% of UB and surpassing the 93.65% and 94.12% achieved by SoftCoT++ at N=10 and $N = 100$, respectively. Meanwhile, removing L_{align} and L_{evo} results in performance drops to 92.65% and 92.87%, respectively; models retaining only SFT or utilizing a Linear-Assistant reach only 92.95%, both underperforming the full model. This indicates that C-Thinker gains do not originate from a single component but from synergy between bidirectional state feedback, protocol generation architecture, and joint optimization objectives.

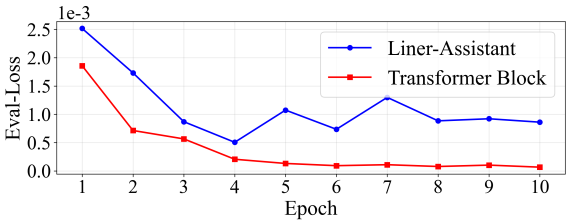
Method	Tibetan-CoT (%)	GSM8K-ZH (%)	GSM8K (%)
SFT-only	29.09	85.97	92.95
w/o L_{evo}	30.00	87.11	92.87
w/o L_{align}	29.09	87.11	92.65
Linear-Ass	29.09	87.19	92.95
CT-UB	35.45	86.81	93.25
CT	30.91	87.64	93.63

Table 4: Comparison results of C-Thinker on cross-lingual mathematical reasoning benchmarks. (N=1).

Table 4 indicates that the aforementioned conclusions generally hold across cross-lingual and cross-distribution scenarios. The full model achieves 87.64% on GSM8K-ZH and 93.63% on the original GSM8K, representing improvements of 0.83 and 0.38 percentage points over C-Thinker UB, respectively. It also consistently outperforms degraded variants such as SFT-only, w/o L_{align}, w/o L_{evo}, and Linear-Assistant. Regarding the more challenging **Tibetan-CoT**, although the current full model has not yet surpassed the UB, a possible reason is that the Primary Model's bottom-layer states contain more noise related to surface-level language, making it harder for the Auxiliary Model to extract stable reasoning cues from feedback. Nevertheless, its accuracy of 30.91% is significantly higher than the 1.80% and 14.59% reported for Qwen2.5-7B under direct answering and explicit Chain-of-Thought settings in the original Ti-MMLU paper, and it also exceeds the 13.94%



(a) Impact of the Number of Latent Protocol Tokens



(b) Eval-Loss during training

Figure 3: Curve Chart of Latent Thought Tokens number (Subfigure a above), and Validation Loss Curves of Different Auxiliary Models (Subfigure b below).

and 18.56% achieved by Qwen2.5-32B. This suggests that while C-Thinker provides steady gains in cross-lingual scenarios with relatively close distributions, it also demonstrates a much stronger capacity for cross-lingual logic transfer compared to original strong baselines in lower-resource Tibetan reasoning tasks with larger distribution gaps, though its stability still has room for improvement.

4.4 Training Dynamics and Protocol Length

We further examine the validation loss curves of the Auxiliary Model and the impact of Latent Protocol length on final accuracy during testing. This provides evidence for the architectural choices and protocol length settings presented in Table 3.

Figure 3a illustrates the GSM8K accuracy for different Latent Protocol lengths. As the number of latent thought tokens increases from 1 to 4, the accuracy improves from 92.80% to 93.63%. However, when the length further increases to 5–8, the results fluctuate slightly between 93.25% and 93.56%, remaining generally lower than the length-4 setting. This trend suggests that the impact of protocol length is most significant in the shorter range: when there are too few tokens, the Auxiliary Model lacks sufficient space to carry intermediate reasoning information. Once the length reaches a certain scale, adding more tokens yields no further gains. This indicates that the performance bottleneck is no longer protocol capacity, but rather the quality and alignment of the protocol content itself. Based on these results, we adopt a fixed Latent Protocol length of 4 for all experiments.

Figure 3b shows validation loss curves for two types of Auxiliary Models. Both exhibit a significant decline during the first 4 epochs, indicating that both linear and Transformer architectures can learn basic protocol generation mappings early in training. However, starting from the 5th epoch, the difference between the two becomes more pronounced: the loss for the Transformer Block remains consistently lower with minimal late-stage fluctuations, while the Linear-Assistant shows noticeable spikes and stays higher. Combined with Table 3, this suggests difference is more than numerical; it reflects a performance gap: 92.95% for the Linear-Assistant versus 93.63% for the full model on GSM8K. A straightforward explanation is that while a linear structure can handle coarse-grained feature mapping, its expressiveness is insufficient for utilizing Primary Model feedback to generate stable protocol representations. By adding a lightweight Transformer Block, the Auxiliary Model integrates feedback more effectively, leading to stable late-stage convergence.

Combining results of the two experiments above, we draw two empirical conclusions: First, the effectiveness of the Auxiliary Model does not depend solely on protocol generation capability; rather, its ability to learn from feedback signals directly determines performance. Second, Latent Protocol length is not "the longer, the better." In our current setup, 4 tokens are sufficient to capture primary performance gains, with marginal returns diminishing significantly as length further increases.

5 Conclusion

We propose **C-Thinker**, a dual-agent reasoning framework based on Latent Protocol Co-evolution. By incorporating hidden states of $\mathcal{M}_R$ as feedback, approach extends latent-space reasoning from unidirectional injection to bidirectional interaction between Primary and Auxiliary Models. Combined with Semantic Consistency Regularization and Intrinsic Evolution objectives, C-Thinker enhances alignment and stability of Latent Protocol. Experimental results show that C-Thinker achieves competitive performance across reasoning benchmarks and exhibits promising logical transfer in low-resource cross-lingual scenarios. Ablation studies confirm that the state feedback path, joint learning objectives, and sufficiently expressive Auxiliary Model are key factors for stable performance gains.

Limitations

Although C-Thinker achieves competitive results on multiple challenging reasoning benchmarks, this work still has several limitations. First, our experiments are conducted primarily on Qwen2.5-7B and Qwen3-8B, and the tasks are mainly limited to mathematics, symbolic reasoning, multiple-choice question answering, and a small set of cross-lingual benchmarks. As a result, the generalization of the framework to larger models, different architectures, and more open-ended long-form generation scenarios remains to be further validated. Second, while we focus on the effects of state feedback, the joint learning objective, and the $\mathcal{M}_T$ architecture on performance, Latent Protocol itself is still a continuous latent representation, and its internal semantics and decision-making process remain only partially interpretable. Finally, although C-Thinker already offers a favorable reasoning cost-performance trade-off with fewer samples, the introduction of an $\mathcal{M}_T$ and additional Latent Protocol tokens still incurs non-negligible training and inference overhead. These limitations do not affect the main conclusion that bidirectional Latent Protocol learning is effective, but they warrant further systematic investigation in future work.

Acknowledgments

Acknowledgments omitted for anonymous review.

References

Meng Cao, Lei Shu, Lei Yu, Yun Zhu, Nevan Wichers, Yinxiao Liu, and Lei Meng. 2024. Enhancing reinforcement learning with dense rewards from language model critic. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 9119–9138.

Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, Jérémy Scheurer, Javier Rando, Rachel Freedman, Tomasz Korbak, David Lindner, Pedro Freire, and 1 others. 2023. Open problems and fundamental limitations of reinforcement learning from human feedback. *arXiv preprint arXiv:2307.15217*.

Jiaqi Chen, Bang Zhang, Ruotian Ma, Peisong Wang, Xiaodan Liang, Zhaopeng Tu, Xiaolong Li, and Kwan-Yee K Wong. 2025. Spc: Evolving self-play critic via adversarial games for llm reasoning. *arXiv preprint arXiv:2504.19162*.

Jeffrey Cheng and Benjamin Van Durme. 2024. Compressed chain of thought: Efficient reasoning through dense representations. *arXiv preprint arXiv:2412.13171*.

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. *Preprint*, arXiv:2110.14168.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, and et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric Xing, and Zhiting Hu. 2022. Rlprompt: Optimizing discrete text prompts with reinforcement learning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3369–3391.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, , and et al. 2024. The llama 3 herd of models. *CoRR*, abs/2407.21783.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. 2025. Group-in-group policy optimization for llm agent training. *arXiv preprint arXiv:2505.10978*.

Fan Gao, Cheng Huang, Yutong Liu, Nyima Tashi, Xiangxiang Wang, Thupten Tsering, Ban Ma-bao, Renzeng Duojie, Gadeng Luosang, Rinchen Dongrub, Dorje Tashi, Xiao Feng Cd, Yongbin Yu, and Hao Wang. 2025a. TLUE: A Tibetan language understanding evaluation benchmark. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 35071–35097, Suzhou, China. Association for Computational Linguistics.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, and 1 others. 2025b. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*.

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. *Transactions of the Association for Computational Linguistics*, 9:346–361.

Prasoon Goyal, Scott Niekum, and Raymond J Mooney. 2019. Using natural language for reward shaping in reinforcement learning. *arXiv preprint arXiv:1903.02020*.

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. *arXiv preprint arXiv:2412.06769*.

Shuo He, Lang Feng, Qi Wei, Xin Cheng, Lei Feng, and Bo An. 2026. Hierarchy-of-groups policy optimization for long-horizon agentic tasks. *arXiv preprint arXiv:2602.22817*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Jian Hu. 2025. Reinforce++: A simple and efficient approach for aligning large language models. *arXiv e-prints*, pages arXiv–2501.

Woosung Koh, Wonbeen Oh, Jaein Jang, MinHyung Lee, Hyeongjin Kim, Ah Yeon Kim, Joonkee Kim, Junghyun Lee, Taehyeon Kim, and Se-Young Yun. 2025. Adastar: Adaptive data sampling for training self-taught reasoners. *arXiv preprint arXiv:2505.16322*.

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 (volume 1: Long papers)*, pages 158–167.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. 2024. Agentsquare: Automatic llm agent search in modular design space. *arXiv preprint arXiv:2410.06153*.

Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. 2025. Codi: Compressing chain-of-thought into continuous space via self-distillation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 677–693.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, and 1 others. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on machine learning research*.

Gemini Team, Rohan Anil, Sebastian Borgeaud, and et al. 2025a. Gemini: A family of highly capable multimodal models. *Preprint*, arXiv:2312.11805.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, and et al. 2025b. Kimi k1.5: Scaling reinforcement learning with llms. *CoRR*, abs/2501.12599.

Ziyu Wan, Yunxiang Li, Xiaoyu Wen, Yan Song, Hanjing Wang, Linyi Yang, Mark Schmidt, Jun Wang, Weinan Zhang, Shuyue Hu, and 1 others. 2025. Rema: Learning to meta-think for llms with multi-agent reinforcement learning. *arXiv preprint arXiv:2503.09501*.

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)*, pages 23336–23351.

An Yang, Anfeng Li, Baosong Yang, and et al. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

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

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. 2024. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*.

Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, and et al. 2024. Chatglm: A family of large language models from GLM-130B to GLM-4 all tools. *CoRR*, abs/2406.12793.

Guibin Zhang, Fanci Meng, Guancheng Wan, Zherui Li, Kun Wang, Zhenfei Yin, Lei Bai, and Shuicheng Yan. 2025a. Latentevolve: Self-evolving test-time scaling in latent space. *arXiv preprint arXiv:2509.24771*.

Junzi Zhang, Jongho Kim, Brendan O'Donoghue, and Stephen Boyd. 2021. Sample efficient reinforcement learning with reinforce. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 10887–10895.

Qiyuan Zhang, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Zhihan Guo, Yufei Wang, Irwin King, Xue Liu, and Chen Ma. 2025b. What, how, where, and how well? A survey on test-time scaling in large language models. *CoRR*, abs/2503.24235.

Yifan Zhang and Team Math-AI. 2024. American invitational mathematics examination (aime) 2024.

Yifan Zhang and Team Math-AI. 2025. American invitational mathematics examination (aime) 2025.

Wanjia Zhao, Mert Yuksekgonul, Shirley Wu, and James Zou. 2025. Sirius: Self-improving multi-agent systems via bootstrapped reasoning. *arXiv preprint arXiv:2502.04780*.

A Appendix

A.1 Training Pseudocode

The training procedure of C-Thinker is summarized below.

Algorithm 1 Training Procedure of C-Thinker

Require: Frozen $\mathcal{M}_R$, trainable $\mathcal{M}_T$, projector $\mathcal{P}_\theta$, dataset $\mathcal{D}$
Ensure: Optimized $\mathcal{M}_T, \mathcal{P}_\theta$
for each batch $(x, y, \text{CoT}) \in \mathcal{D}$ **do**
  $\mathbf{H}_F \leftarrow \text{Layer}_1(\mathcal{M}_R, x)$ ▷ Extract feedback
  $\mathbf{H}_T \leftarrow \mathcal{M}_T(x, \mathbf{H}_F)$ ▷ C-Attn interaction
  $\mathbf{Z} \leftarrow \mathcal{P}_\theta(\mathbf{H}_T)$ ▷ Generate protocol
  $L_{\text{SFT}} \leftarrow -\sum_t m_t \log p_{\mathcal{M}_R}(y_t \mid x, \mathbf{Z}, y_{<t})$
  $\mathbf{h}_q \leftarrow \text{Proj}(x_{\text{ques}})$
  $L_{\text{align}} \leftarrow \frac{1}{L}\sum_{j=1}^{L} D_{\text{KL}}(\sigma(\mathbf{h}_q) \parallel \sigma(\mathbf{z}_j))$
  $\{\mathbf{s}_k\} \leftarrow \text{Encode}(\text{CoT}),\ \mathbf{a} \leftarrow \text{Proj}(y)$
  $k^\star \leftarrow \arg\max_k \cos(\mathbf{s}_k, \mathbf{a})$
  $L_{\text{evo}} \leftarrow -\log \pi_\phi(k^\star \mid x)$
  $L_{\text{total}} \leftarrow \lambda_{\text{sft}} L_{\text{SFT}} + \lambda_{\text{align}} L_{\text{align}} + \lambda_{\text{evo}} L_{\text{evo}}$
  Update $\{\mathcal{M}_T, \mathcal{P}_\theta\}$ via ∇L_{total}
end for

A.2 Sizes of Datasets

We report below the sizes of the training and test splits for all datasets used in our experiments. Following the main experimental setup, we omit the development split here and only describe the training and test data sizes.

All datasets and pretrained models used in this work are publicly available and used in accordance with their original licenses and terms of use.

The Tibetan-CoT dataset will be released for research purposes only. Potential risks also remain. Since Tibetan-CoT relies on AI-generated rationales, the augmented explanations may contain factual errors, linguistic artifacts, or culturally biased reasoning patterns, especially under low-resource Tibetan settings where automatic quality control is less reliable. In addition, the Latent Protocol is a continuous representation and is not fully interpretable, so erroneous protocol guidance may be difficult to diagnose and may lead to overconfident incorrect answers. C-Thinker is intended for research evaluation of reasoning models rather than direct deployment in high-stakes decision-making scenarios.

- **GSM8K**: The training set contains 7,473 samples, and the test set contains 1,319 samples.
- **GSM8K-ZH**: The training set contains 7,471 samples, and the test set contains 1,319 samples.
- **AQuA**: The training set contains 97,467 samples, and the test set contains 254 samples.
- **StrategyQA**: The training set contains 1,832 samples, and the test set contains 458 samples.
- **Date Understanding (DU)**: The dataset contains 369 samples for testing.
- **MATH**: The training set contains 7,496 samples, and the test set contains 500 samples.
- **AIME24**: The dataset contains 30 samples. In the current setup, the same 30-sample split is used for testing.
- **AIME25**: The dataset contains 30 samples. In the current setup, the same 30-sample split is used for testing.
- **Tibetan-CoT**: The training set contains 2,200 samples, and the test set contains 110 samples.

Due to the limited sample sizes of DU, AIME24, and AIME25, we evaluate DU using the checkpoint trained on GSM8K, and evaluate AIME24 and AIME25 using the checkpoint trained on MATH.

A.3 Prompts for Reasoning Augmentation on the Tibetan CoT Dataset

We design five prompt variants for Tibetan multiple-choice rationale augmentation, each emphasizing a distinct reasoning pattern: core concept matching, distractor elimination, condition decomposition, rule/formula application, and teacher-style summarization.

[TIBETAN_FINAL_ANSWER_LINE] denotes a fixed Tibetan answer template meaning "The answer is [CORRECT_ANSWER]."

Prompt Type I: Core Concept Matching

```
You are a data construction assistant
↪  skilled at generating Tibetan rationales
for multiple-choice questions. I will
↪  provide a multiple-choice question, its
options, and the correct answer. Please
↪  generate only the rationale for this
sample.

Requirements:
1. Output in Tibetan and keep it concise and
↪  clear.
2. Write a total of 3-4 lines, with each
↪  line representing one step of reasoning.
3. In the first line, identify the core
↪  concept or keyword being tested in the
   question.
4. In the second line, explain what this
↪  core concept means in the context of
   the current question.
5. In the third line, state which option
↪  best matches this concept.
```

```
6. The final line must be exactly:
↪  [TIBETAN_FINAL_ANSWER_LINE]
7. Do not output anything other than the
↪  rationale itself.

Question:
[QUESTION]

Options:
[OPTIONS]

Correct Answer:
[CORRECT_ANSWER]
"""
```

Prompt Type II: Elimination of Distractors

```
You are an assistant skilled at generating
↪  Tibetan reasoning processes for
multiple-choice questions. Based on the
↪  given question, options, and correct
answer, write a rationale that mainly uses
↪  elimination.

Output requirements:
1. Output entirely in Tibetan.
2. Write a total of 4-5 lines, with each
↪  line as an independent reasoning step.
3. First, use one line to explain the key
↪  basis for judging this question.
4. In the middle 2-3 lines, briefly explain
↪  why several incorrect options do
   not fit the question. You may eliminate
   ↪  them in groups rather than discussing
   each one evenly.
5. In the second-to-last line, explain why
↪  the remaining correct option is the
   most appropriate.
6. The final line must be exactly:
↪  [TIBETAN_FINAL_ANSWER_LINE]
7. Do not use expressions such as "I think"
↪  or "according to the correct answer."
8. Output only the rationale body.

Question:
[QUESTION]

Options:
[OPTIONS]

Correct Answer:
[CORRECT_ANSWER]
"""
```

Prompt Type III: Condition Decomposition and Verification

```
You are an assistant for generating Tibetan
↪  question analyses. Please write the
reasoning process for the following
↪  multiple-choice question in a "condition
decomposition" style, and output only the
↪  rationale field content.

Requirements:
1. Output in Tibetan.
2. Write around 4 lines in total, with each
↪  line representing one step of
   reasoning.
3. In the first line, break the question
↪  down into 2-3 key conditions or
   judgment points.
4. In the second line, explain the judgment
↪  results corresponding to the first
   two conditions.
5. In the third line, state which option
↪  satisfies these conditions at the same
   time.
6. The final line must be exactly:
↪  [TIBETAN_FINAL_ANSWER_LINE]
7. Do not copy the whole question verbatim;
↪  summarize it concisely.
8. Do not output any extra explanation.

Question:
[QUESTION]

Options:
[OPTIONS]

Correct Answer:
[CORRECT_ANSWER]
"""
```

Prompt Type IV: Rule / Formula Application

```
You are an assistant skilled at generating
↪  step-by-step Tibetan reasoning for
multiple-choice questions. Based on the
↪  question, options, and correct answer,
write a rationale that first invokes a rule
↪  and then applies it to the current
problem.

Output requirements:
1. Output entirely in Tibetan.
2. Write a total of 3-5 lines, with each
↪  line being a small reasoning step.
3. In the first line, point out the formula,
↪  rule, principle, or judgment
   criterion needed for this question.
4. In the second line, apply this rule to
↪  the information in the question. If
   numbers are involved, calculate them
   ↪  directly; if not, apply the rule to
   ↪  the
   meaning of the question.
5. In the third line, explain which option
↪  best fits on this basis.
6. The final line must be exactly:
↪  [TIBETAN_FINAL_ANSWER_LINE]
7. If calculation is involved, keep the
↪  process brief and do not expand it into
   a long explanation.
8. Output only the rationale, and do not
↪  repeat the question or options.

Question:
[QUESTION]

Options:
```

```
[OPTIONS]

Correct Answer:
[CORRECT_ANSWER]
"""
```

Prompt Type V: Teacher-Style Explanation and Summarization

```
You are now playing the role of a teacher
↪  explaining multiple-choice questions
in Tibetan. Based on the given question,
↪  options, and correct answer, write a
natural, clear, and concise step-by-step
↪  rationale, and output only the
rationale.

Requirements:
1. Output in Tibetan, with a teacher-like
↪  tone, but do not make it overly
   colloquial.
2. Write a total of 4-5 lines, with each
↪  line forming an independent segment of
   a continuous reasoning process.
3. In the first line, summarize the most
↪  important reasoning idea for this
   question.
4. In the second and third lines, explain
↪  why the correct option fits the
   question, and you may briefly compare it
   ↪  with 1-2 distractors.
5. In the second-to-last line, summarize the
↪  judgment result in one sentence.
6. The final line must be exactly:
↪  [TIBETAN_FINAL_ANSWER_LINE]
7. Do not output numbering, titles, notes,
↪  or extra symbols.

Question:
[QUESTION]

Options:
[OPTIONS]

Correct Answer:
[CORRECT_ANSWER]
"""
```

A.4 Prompt Templates for Training and Evaluation

This subsection reports the textual prompt templates used for baseline methods and for decoding the Primary Model during evaluation. These templates should not be interpreted as the mechanism by which C-Thinker generates its Latent Protocol. In C-Thinker, $\mathcal{M}_T$ produces continuous protocol vectors that are injected into reserved latent slots of $\mathcal{M}_R$, rather than generating textual `soft_thoughts`.

In this appendix, we retain only the prompts used by Qwen2.5 on GSM8K and AIME. Virtually identical prompts are used for both AIME24 and AIME25. The appendix currently lists the GSM8K and AIME24 prompts.

During training, we remove rationale traces that directly contain the final answer from the training data. During evaluation, no rationale traces from the test set are provided.

Base Model Prompt for GSM8K

```
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}

Please think step by step and provide a
↪  detailed reasoning process. 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_thoughts}
"""
```

Assistant Prompt for GSM8K

```
You are required to generate
↪  {num_thought_tokens} 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.
```

```
- The other language model is good enough to
↪  understand the problem, so what
  you need to do is generate some
  ↪  informative key tokens that summarize
  ↪  the
  problem.
...

Here is the problem: {question}.
"""
```

Base Model Prompt for AIME24

```
You are an expert competition mathematician.
Solve the following AIME problem
↪  efficiently, carefully, and clearly:
- Verify intermediate results and avoid
↪  algebra mistakes.
- If multiple methods exist, choose the most
↪  reliable one.
- Double-check the final result.
- For simple problems (2 steps or fewer):
Provide a concise solution with minimal
↪  equation, but still show the key
reasoning steps.
- 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 the final integer from 0
↪  to 999 that solves the problem.
Do not provide the final answer outside of
↪  \boxed{}.

Problem: {question}

Please think step by step and provide a
↪  detailed reasoning process. Verify
intermediate results, avoid algebra
↪  mistakes, and double-check the final
result. 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_thoughts}
"""
```

Assistant Prompt for AIME24

```
You are required to generate
↪  {num_thought_tokens} tokens to help
↪  another
language model to solve the following AIME
↪  2024 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.
- The other language model must place its
↪  final number or expression within
  \boxed{}.
- The other language model is good enough to
↪  understand the problem, so what
  you need to do is generate some
  ↪  informative key tokens that summarize
  ↪  the
  problem.
...

Here is the problem: {question}.
"""
```

A.5 Hyperparameters.

For all training experiments, we set the learning rate to 2×10^{-5}, the warmup ratio to 0.1, and the number of training epochs to 16–32, depending on the dataset scale and convergence behavior. The default per-device batch size is set to 2 with gradient accumulation steps of 8; for datasets with longer inputs and higher memory costs, we use a per-device batch size of 1 and increase the gradient accumulation steps to 16 accordingly.

To avoid additional dataset-specific hyperparameter tuning and ensure comparability across datasets, we adopt a unified global setting: we use 32 latent protocols during training to improve optimization stability and robustness, and use 4 latent protocols during evaluation to control inference cost while maintaining efficiency.

During training, the parameters of the main language model remain frozen. The trainable components mainly include the randomly initialized lightweight reasoning module $\mathcal{M}_T$ and the projection layer, so that parameter updates are concentrated on the protocol modeling and representation alignment components of C-Thinker.

The maximum input length follows the current code implementation, and by default no additional global truncation limit is imposed.

During evaluation, we use `do_sample=True`. For Qwen-series models, we uniformly set

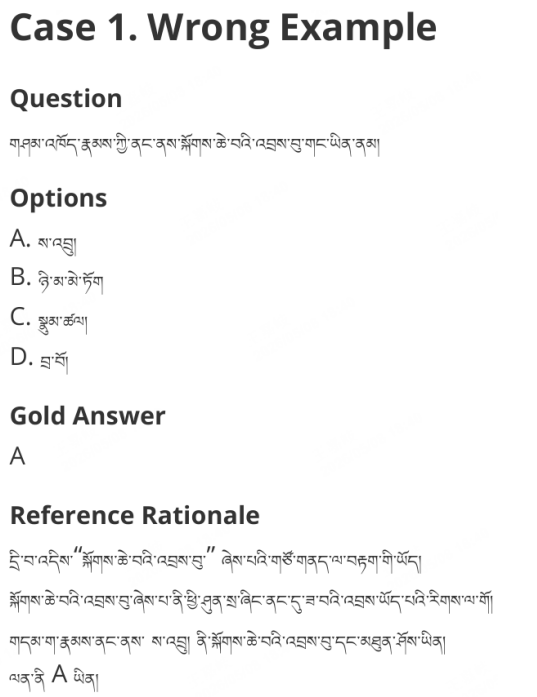


Figure 4: Original Tibetan question for Case 1 (failure case).

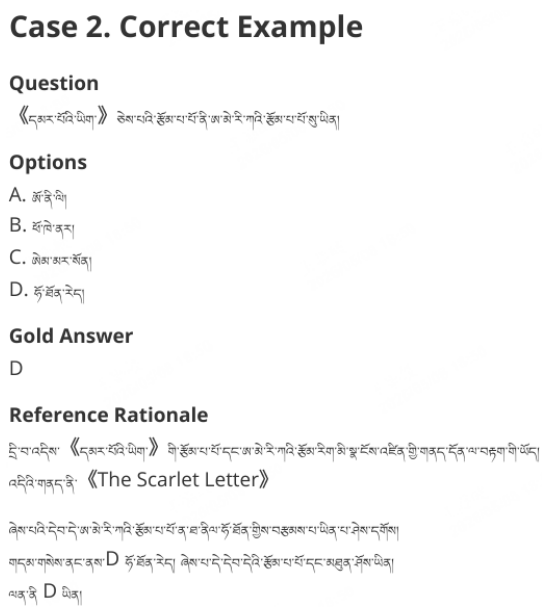


Figure 5: Original Tibetan question for Case 2 (success case).

`temperature=0.7` and `top_p=0.8`. Here, N denotes the number of sampled candidate outputs for each test example: when N=1, we report the accuracy of a single sampled output; when N=10, we first sample 10 candidate outputs and then apply majority voting over the extracted final answers to compute the final accuracy. The C-Thinker results reported in the paper are the average of the best results over four random seeds.

Experiments were implemented using the HuggingFace Transformers library and PyTorch with default settings unless otherwise specified.

AI assistants were used for language polishing and minor writing assistance. All technical content, experiments, and conclusions were verified by the authors.

A.6 Case Study on Tibetan-CoT

To provide a more concrete view of model behavior on Tibetan-CoT, we present two representative test examples: one failure case and one success case. We show the original Tibetan questions as images, while the English translations are provided as plain text. The first case corresponds to test sample #1, and the second corresponds to test sample #8.

Case 1: Failure Case. Figure 4 shows the original Tibetan question.
English translation.

> **Question:** Which of the following is a hard-shelled fruit (i.e., a nut)?
> **Options:** (A) Peanut; (B) Sunflower; (C) Rapeseed; (D) Sorghum.
> **Gold answer:** A.
> **Reference rationale:** This question tests the core concept of a nut. A nut is a type of fruit with a hard outer shell and an edible part inside. Among the options, peanut best matches this definition. Therefore, the answer is A.

Case 2: Success Case. Figure 5 shows the original Tibetan question.
English translation.

> **Question:** Which American writer is the author of *The Scarlet Letter*?
> **Options:** (A) O'Neill; (B) Faulkner; (C) Emerson; (D) Hawthorne.
> **Gold answer:** D.
> **Reference rationale:** This question tests knowledge of the author of *The Scarlet Letter* and basic identification of American literary figures. The key point is to know that *The Scarlet Letter* was written by Nathaniel Hawthorne. Among the options, Hawthorne best matches the correct author. Therefore, the answer is D.