Incorporating Explicit and Implicit Chain-of-Thought as Hybrid Reasoning Paradigm with Guided Latent Space Modeling

Guangyu Li[1,2,3,*], Zhekai Ge[1,2], Zhuo Chen[1,2,3], Zhen Zhang[3], Xinyu Wang[3], Yong Jiang[3], Kewei Tu[1,2,†]

[1]School of Information Science and Technology, ShanghaiTech University

[2]Shanghai Engineering Research Center of Intelligent Vision and Imaging

{ligyu,gezhk2024,chenzhuo,tukw}@shanghaitech.edu.cn

[3]Tongyi Lab, Alibaba Group

{zhzhen33,wangxinyu.nlp,jiangyong.ml}@gmail.com

Abstract

Chain-of-Thought (CoT) has proven effective for eliciting reasoning abilities in large language models, but standard autoregressive decoders treat reasoning and answer tokens uniformly as next-token prediction targets, despite their fundamentally different roles. This homogeneous modeling introduces conflicting inductive biases and significant inference overhead. Latent CoT methods attempt to improve efficiency by removing explicit reasoning tokens, yet often degrade performance, indicating that explicit reasoning supervision remains critical for preserving pretrained reasoning capabilities. To address these challenges, we propose Pre^2, a phase-aware reasoning framework that decouples thinking and answering while retaining explicit CoT supervision. Our approach introduces a guided latent space and predicts independent pre-thinking and pre-answering latent states to guide the model in generating reasoning tokens and answering tokens. We integrate them with explicit token representations through a hybrid latent–token decoding paradigm. This design increases computation along both width and depth dimensions with minimal parameter overhead, enabling stronger reasoning without sacrificing inference efficiency. Training is conducted exclusively on one in-domain dataset, with direct zero-shot evaluation on six out-of-domain benchmarks, and experiments demonstrate consistent and significant improvements over strong baselines and robust generalization.

1 Introduction

Chain-of-Thought (CoT) prompting, introduced by Wei et al. (2022), has proven effective in enhancing the reasoning abilities of large language models by inducing intermediate reasoning steps, yielding strong gains on mathematical and multi-step reasoning tasks (Guo et al., 2025; OpenAI, 2024; Muennighoff et al., 2025). However, these prompt-based methods rely on explicitly generating reasoning tokens at inference time (Wang et al., 2022; Zhou et al., 2022; Kojima et al., 2022; Zhang et al., 2022). Due to the autoregressive nature of language models, early reasoning errors can propagate through subsequent predictions, often resulting in brittle and unstable reasoning trajectories.

To mitigate this limitation, a line of work has explored test-time scaling strategies that aggregate or refine reasoning paths through increased computation at inference. Representative approaches include self-consistency (Wang et al., 2022), which samples multiple reasoning trajectories and selects the most consistent answer, as well as more structured methods such as Tree-of-Thoughts (ToT, Yao et al. 2023), which explicitly searches over reasoning branches. While effective, these methods significantly increase inference cost, as they require extensive sampling, branching, or search procedures, making them impractical for latency- or budget-constrained settings (Li et al., 2024; Wang et al., 2025).

More recently, latent CoT approaches have emerged as an alternative direction, aiming to bypass explicit reasoning token generation altogether (Wei et al., 2025; He et al., 2025; Liu and Zhang, 2025). Instead of producing natural language reasoning steps, these methods perform reasoning directly in a continuous latent space, thereby reducing inference-time token generation. Representative works such as Coconut (Hao et al., 2024), CoDi (Shen et al., 2025), and SoftCoT (Xu et al., 2025) demonstrate that latent representations can capture certain aspects of reasoning while improving efficiency. Nevertheless, by skipping or discarding explicit reasoning tokens, these approaches often struggle to fully preserve the reasoning capabilities of pretrained models, highlighting a fundamental trade-off between efficiency and reasoning fidelity

[*]This work was conducted when Guangyu Li and Zhuo Chen were interning at Tongyi Lab.

[†]Corresponding author.

(Xia et al., 2025; Liu et al., 2025).

Traditional methods uniformly model reasoning and answer tokens, ignoring their fundamentally different patterns (Nye et al., 2021; Lampinen et al., 2022). While latent reasoning approaches eliminate CoT for efficiency, we argue that CoT provides essential supervision signals aligned with pretraining. Fully eliminating CoT may weaken reasoning ability (Turpin et al., 2023). To address these limitations, we explicitly decouple the modeling of reasoning and answering by introducing independent pre-thinking and pre-answering phases, and our approach preserves explicit reasoning supervision while introducing structured latent phases, balancing inference efficiency and reasoning expressivity. Specifically, instead of forcing the model to autoregressively commit to explicit reasoning tokens, we supervise continuous latent representations that summarize the reasoning-ready and answer-ready states. This separation enables the model to first internalize a structured reasoning abstraction before transitioning into answer generation, thereby strengthening its reasoning capability while reducing reliance on verbose or brittle CoT generation.

We introduce Pre^2, a novel framework that effectively addresses all of the above-mentioned problems simultaneously. We model Chain-of-Thought reasoning as a structured latent trajectory composed of a reasoning-ready state and an answer-ready state. Instead of autoregressively generating reasoning tokens, we supervise continuous latent representations aligned with pooled reasoning and answer embeddings, enabling non-autoregressive and controllable reasoning abstraction. Our contributions are summarized as follows: (1) We propose a phase-aware reasoning framework that independently models the pre-thinking and pre-answering processes, effectively decoupling the conflicting patterns between thinking and answering stages. (2) We introduce a hybrid paradigm that combines latent representations with explicit token-level supervision, requiring only a minimal increase in model parameters while preserving strong performance and high inference efficiency. (3) We conduct extensive experiments on one in-domain dataset and six out-of-domain benchmarks, where our method consistently outperforms strong baselines, demonstrating robust generalization across domains.

2 Related Works

Explicit Chain-of-Thought Reasoning. Pioneering works demonstrate that explicitly generating intermediate reasoning steps significantly boosts performance on arithmetic and symbolic tasks (Wei et al., 2022; Kojima et al., 2022). Despite the successes, explicit CoT suffers from inherent inefficiency due to the autoregressive generation of long reasoning sequences. Furthermore, treating reasoning tokens and answer tokens as homogeneous prediction targets introduces structural conflicts; reasoning trajectories often require exploratory and verbose patterns, whereas answers demand conciseness (Nye et al., 2021; Lampinen et al., 2022).

Latent Chain-of-Thought Reasoning. To mitigate the inference overhead of explicit tokens, Latent CoT approaches aim to internalize reasoning into continuous hidden states or implicit processes, enabling models to "think before speaking" (Zelikman et al., 2024). Methods like implicit reasoning via stepwise knowledge internalization (Deng et al., 2024) and teacher-student distillation (e.g., CODI, Shen et al. 2025) attempt to transfer reasoning semantics from explicit traces to model embeddings. Other approaches, such as Coconut (Hao et al., 2024) and CCoT (Cheng and Van Durme, 2024), explore reasoning in continuous space by compressing steps into dense vectors or "contemplation tokens". However, purely latent methods face significant challenges: they often suffer from performance degradation on complex tasks due to the lack of fine-grained supervision signals (Hao et al., 2024). Unlike explicit CoT, where every step is verified, methods like Coconut typically rely on answer-level supervision, while distillation methods like CODI focus on trajectory-level alignment, potentially losing critical step-wise logic.

Computational Scaling across Width and Depth Dimensions. Existing computational scaling methods face limitations: width-based approaches (e.g., Pause Tokens, Goyal et al. 2023) often lack explicit semantic guidance, while depth-based methods (e.g., MoR, Bae et al. 2025 and Ouro Zhu et al. 2025) keep reasoning implicit within recursive loops. We bridge these dimensions by scaling width via learnable latent tokens and depth through independent modules.

Our approach leverages a hybrid framework combining explicit and latent Chain-of-Thought reasoning. Unlike pause tokens that operate exclusively with static token-level embeddings, our

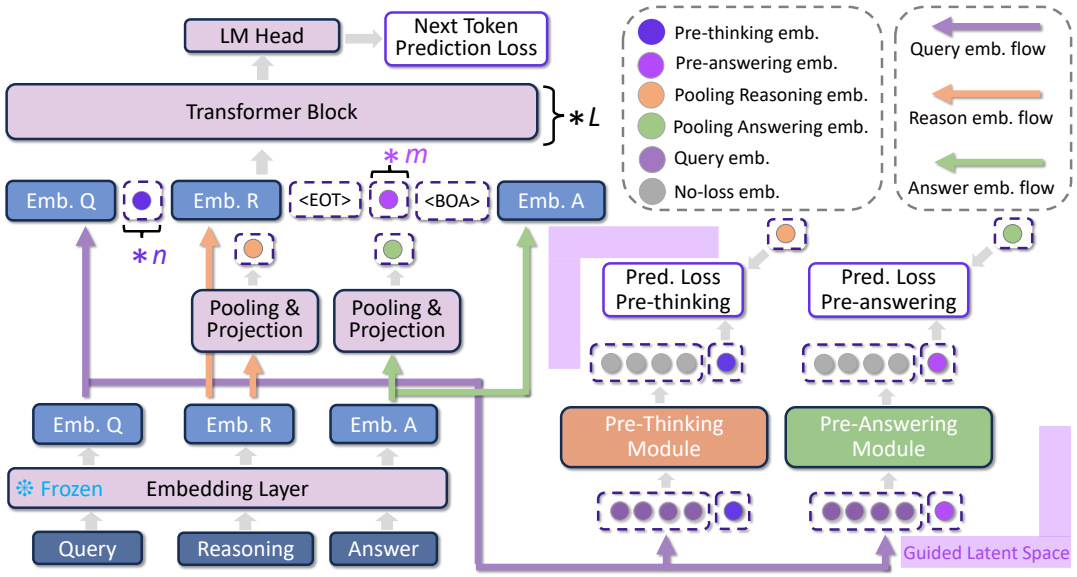

Figure 1: We froze the embedding layer and adopt pooled reasoning embedding and answering embedding as the supervision signal. Then the predicted pre-thinking embeddings and pre-answering emb. from the Guided Latent Space (The bottom right part.) will be fed into the Transformer block together with all explicit embeddings

framework employs learnable sentence-level embeddings to capture sequence representations of complete reasoning steps and answer tokens.

3 Methodology

3.1 Preliminary of Pre[2]

Let $\mathcal{V}$ denote a vocabulary of size $V = |\mathcal{V}|$. Given a training sample, we define a CoT sequence as

$$\mathbf{X} = (\mathbf{Q} := x^{(q)},\ \mathbf{R} := x^{(r)},\ \mathbf{A} := x^{(a)}),$$

where $x^{(q)} = (q_1, \ldots, q_{L_q}) \in \mathcal{V}^{L_q}$, $x^{(r)} = (r_1, \ldots, r_{L_r}) \in \mathcal{V}^{L_r}$ and $x^{(a)} = (a_1, \ldots, a_{L_a}) \in \mathcal{V}^{L_a}$ denote the query, reasoning steps, and answer.

Our framework introduces a novel approach where three built-in embeddings serve as initialization for the latent states of pre-thinking and pre-answering. We introduce three continuous latent vectors that have been initialized within our model:

$$\mathbf{z}^{(0)}_{\text{think}} \in \mathbb{R}^d, \mathbf{z}^{(0)}_{\text{ans}} \in \mathbb{R}^d, \mathbf{z}^{(0)}_{\text{boa}} \in \mathbb{R}^d,$$

where $\mathbf{z}^{(0)}_{\text{think}}$, $\mathbf{z}^{(0)}_{\text{ans}}$ and $\mathbf{z}^{(0)}_{\text{boa}}$ are learnable global parameters. Specifically, $\mathbf{z}^{(0)}_{\text{think}}$ is trained to learn a latent representation of the complete reasoning process; $\mathbf{z}^{(0)}_{\text{ans}}$ captures an abstract representation of the answer; $\mathbf{z}^{(0)}_{\text{boa}}$ an answer indicator that signals the model to transition into answer generation mode.

Our model uses a frozen token embedding matrix $\mathcal{E}$ to get different non-contextualized embeddings for query, reasoning and answer tokens. This approach offers an additional advantage: the frozen embedding layer provides stable training targets while preserving model capacity, thus avoiding performance degradation. Let $\mathcal{E} : \mathcal{V} \to \mathbb{R}^d$ be a shared token embedding function. The embedded sequences are given by:

$$\mathbf{Q} = (\mathbf{q}_1, \ldots, \mathbf{q}_{L_q}) \in \mathbb{R}^{L_q \times d},$$
$$\mathbf{R} = (\mathbf{r}_1, \ldots, \mathbf{r}_{L_r}) \in \mathbb{R}^{L_r \times d},$$
$$\mathbf{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_{L_a}) \in \mathbb{R}^{L_a \times d}.$$

We define a pooling operator $\mathcal{P}(\cdot)$ to obtain sequence-level representations. Then the pooled reasoning and answers representations will be regarded as gold latent representations because we assume that these pooled representations include all necessary information to guide the model to generate complete reasoning and answering tokens.

$$\bar{\mathbf{r}} = \mathcal{P}(\mathbf{R}) \in \mathbb{R}^d,$$
$$\bar{\mathbf{a}} = \mathcal{P}(\mathbf{A}) \in \mathbb{R}^d.$$

3.2 Pre-thinking and Pre-answering Modeling

The modeling of pre-thinking and pre-answering is centered on a key principle: we aim to obtain embeddings that encapsulate holistic information. Specifically, **pre-thinking embeddings** contain information about the full reasoning process, and **pre-answering embeddings** contain information about the final answer. With these global representations, the model can provide top-down guidance

and predictions throughout the subsequent autoregressive generation of reasoning tokens and answer tokens, rather than relying solely on local, step-by-step predictions. We now address the key question: how are these compressed embeddings learned?

We make a key empirical observation: for a small proportion of complex reasoning tasks, models can produce correct answers directly from the query even when the generated reasoning steps are unreliable or incorrect. Motivated by this observation, we design our approach to leverage query tokens as conditions, using the built-in embeddings as latent state initializations and employing two lightweight Transformer decoder blocks to causally model token-level dependencies. Moreover, recent studies have demonstrated that increasing per-token computation depth yields unique benefits that cannot be achieved by simply scaling the width of sequence modeling, particularly for certain reasoning tasks (Bae et al., 2025; Zhu et al., 2025). To enhance computational and inference efficiency, we selectively allocate additional depth-wise computation exclusively to reasoning and answer tokens.

Let f_{think} denote the pre-thinking module. The input sequence is constructed as

$$\mathbf{H}^{(0)}_{\text{think}} = \big[\mathbf{Q}; \mathbf{z}^{(0)}_{\text{think}}\big],$$

where $[\cdot;\cdot]$ is concatenation operator.

The hidden states are obtained by

$$\mathbf{H}_{\text{think}} = f_{\text{think}}(\mathbf{H}^{(0)}_{\text{think}}).$$

We take the final hidden state as the predicted pre-thinking latent representation:

$$\hat{\mathbf{z}}_{\text{think}} = \mathbf{H}_{\text{think}}[-1].$$

Similarly, we define a pre-answering module f_{ans} with input

$$\mathbf{H}^{(0)}_{\text{ans}} = \big[\mathbf{Q}; \mathbf{z}^{(0)}_{\text{ans}}\big].$$

The predicted pre-answering latent is given by

$$\hat{\mathbf{z}}_{\text{ans}} = f_{\text{ans}}(\mathbf{H}^{(0)}_{\text{ans}})[-1].$$

Specifically, the two modules consist of two Transformer blocks and utilize a causal mask for attention computation. This design enables both $\mathbf{z}^{(0)}_{\text{think}}$ and $\mathbf{z}^{(0)}_{\text{ans}}$ to attend to the complete set of query token embeddings, allowing modeling pre-thinking and pre-answering separately. To enable $\mathbf{z}^{(0)}_{\text{think}}$ and $\mathbf{z}^{(0)}_{\text{ans}}$ to learn representations that can predict the overall architecture encompassing the thinking and answering stages, we let $\bar{\mathbf{r}}$ and $\bar{\mathbf{a}}$ be the supervised signal for pre-thinking modeling and pre-answering modeling, and the loss function will be either smooth L1 or L2 loss. To prevent gradient backpropagation and ensure unidirectional learning flow, we apply a stop-gradient operation ($\mathcal{SG}(\cdot)$) on $\bar{\mathbf{r}}$ and $\bar{\mathbf{a}}$. Thus, the pre-thinking alignment loss (e.g., L2 norm) is defined as

$$\mathcal{L}_{\text{think}} = \|\hat{\mathbf{z}}_{\text{think}} - \mathcal{SG}(\bar{\mathbf{r}})\|_2^2 . \tag{1}$$

The pre-answering alignment loss is defined as

$$\mathcal{L}_{\text{ans}} = \|\hat{\mathbf{z}}_{\text{ans}} - \mathcal{SG}(\bar{\mathbf{a}})\|_2^2 . \tag{2}$$

3.3 Training of Pre^2

In this section, we describe the training procedure. Let $\mathbf{e}_{\text{eot}} \in \mathbb{R}^d$ denote the end-of-thought embedding. The input sequence is constructed as

$$\mathbf{H}^{(0)}_{\text{dec}} = \big[\mathbf{Q}; \hat{\mathbf{z}}_{\text{think}}; \mathbf{R}; \mathbf{e}_{\text{eot}}; \hat{\mathbf{z}}_{\text{ans}}; \mathbf{z}_{\text{boa}}; \mathbf{A}\big] \in \mathbb{R}^{T_{\text{dec}} \times d},$$

where $T_{\text{dec}} = L_q + 1 + L_r + 1 + 1 + 1 + L_a$.

And the $\mathbf{e}_{\text{eot}}$ is a special token that is already in the vocabulary of the pre-trained backbone model. And this $\mathbf{e}_{\text{eot}}$ is essential for our inference pipeline because it can provide a proper signal that tells us the reasoning stage is completed.

The decoder f_{dec} produces hidden states

$$\mathbf{H}_{\text{dec}} = f_{\text{dec}}(\mathbf{H}^{(0)}_{\text{dec}}).$$

The final supervision labels are defined as

$$\mathbf{Y} = \big[\underbrace{\texttt{-100}}_{\mathbf{Q}},\ \mathbf{R},\ \mathbf{e}_{\text{eot}},\ \underbrace{\texttt{-100}}_{\hat{\mathbf{z}}_{\text{ans}}},\ \underbrace{\texttt{-100}}_{\mathbf{z}_{\text{boa}}},\ \mathbf{A}\big],$$

where `-100` indicates masked positions that do not contribute to the loss.

Let $\mathcal{T}_{\text{sup}} \subset \{1, \ldots, T_{\text{dec}}\}$ denote the set of supervised token positions, given by

$$\mathcal{T}_{\text{sup}} = \mathcal{T}_{\text{cot}} \ \cup\ \mathcal{T}_{\text{ans}},$$

where

$$\mathcal{T}_{\text{cot}} = \{\, t \mid x_t \in \mathbf{R} \cup \{\mathbf{e}_{\text{eot}}\} \,\}, \mathcal{T}_{\text{ans}} = \{\, t \mid x_t \in \mathbf{A} \,\}.$$

The masked language modeling loss is defined as

$$\begin{aligned} \mathcal{L}_{\text{LM}} &= -\sum_{t \in \mathcal{T}_{\text{sup}}} \log p_\theta(y_t \mid \mathbf{H}^{(<t)}_{\text{dec}}) \\ &= -\sum_{t \in \mathcal{T}_{\text{cot}}} \log p_\theta(r_t \mid \mathbf{H}^{(<t)}_{\text{dec}}) \\ &\quad -\sum_{t \in \mathcal{T}_{\text{ans}}} \log p_\theta(a_t \mid \mathbf{H}^{(<t)}_{\text{dec}}). \end{aligned} \tag{3}$$

Algorithm 1 Inference Process of **Pre**^{2}

Require: Token embeddings $\mathbf{x}$ from frozen embedding layer;
Require: Stage-flags: `is_prefilling`, `is_cot_end`, `is_answering`
Ensure: Updated embeddings $\mathbf{x}$

if `is_prefilling` **then**
  $\hat{\mathbf{z}}_{\text{t}} = f_{\text{think}}([x; \mathbf{z}^{(0)}_{\text{think}}])[-1]$
  $\hat{\mathbf{z}}_{\text{a}} = f_{\text{answer}}([x; \mathbf{z}^{(0)}_{\text{answer}}])[-1]$
  $\mathbf{x} \leftarrow [\mathbf{x}; \hat{\mathbf{z}}_{\text{t}}]$
else
  if `is_cot_end` **then**
    if $\neg$`is_answering` **then**
      $\mathbf{x} \leftarrow [\mathbf{q}; \hat{\mathbf{z}}_{\text{t}}; \text{CoT}; \langle\text{eot}\rangle; \hat{\mathbf{z}}_{\text{a}}; \mathbf{z}_{\text{boa}}]$
      `is_answering` $\leftarrow$ `True`
    else
      $\mathbf{x} \leftarrow [\mathbf{q}; \hat{\mathbf{z}}_{\text{t}}; \text{CoT}; \langle\text{eot}\rangle; \hat{\mathbf{z}}_{\text{a}}; \mathbf{z}_{\text{boa}}; \text{Ans}]$
    end if
  else
    $\mathbf{x} \leftarrow [\mathbf{q}; \hat{\mathbf{z}}_{\text{t}}; \text{CoT}]$
  end if
end if
return $\mathbf{x}$

The overall training objective is given by

$$\mathcal{L} = (1-\lambda_{\text{think}}-\lambda_{\text{ans}})\mathcal{L}_{\text{LM}}+\lambda_{\text{think}}\mathcal{L}_{\text{think}}+\lambda_{\text{ans}}\mathcal{L}_{\text{ans}}. \tag{4}$$

3.4 Inference of Pre^2

The inference procedure differs from vanilla autoregressive decoding in several ways. Notably, we must explicitly maintain: (1) the current query length, and (2) an End-of-Thought (EoT) signal to determine when CoT generation concludes and answer generation should begin. We provide a detailed pseudocode description in Algorithm 1, where $\mathbf{q}$ is query embeddings, $\hat{\mathbf{z}}_{\text{t}}$ is predicted pre-thinking embedding, $\hat{\mathbf{z}}_{\text{a}}$ is predicted pre-answering embedding, $\mathbf{z}_{\text{boa}}$ is the answering indicator embedding that initialized in the model. Specifically, during the prefilling phase, we execute the pre-thinking $f_{\text{think}}(\cdot)$and pre-answering modules $f_{\text{ans}}(\cdot)$ to generate $\hat{\mathbf{z}}_{\text{t}}$ and $\hat{\mathbf{z}}_{\text{a}}$ before any token generation begins. In the autoregressive generation stages that follow, we dynamically reconstruct the input sequence $\mathbf{X}$ through concatenation operations, ensuring that the model's behavior remains consistent across training and inference.

4 Experimental Design

4.1 Datasets

We train the Pre^2 on GSM8K-Aug (Deng et al., 2024), which augments the original GSM8K (Cobbe et al., 2021) training set with questions. For evaluation, we assess reasoning robustness and generalization across 7 benchmarks. The in-domain evaluation uses the standard GSM8K (Cobbe et al., 2021) test split, while out-of-domain evaluation employs 6 datasets: GSM-Hard (increased arithmetic difficulty through larger integers) (Gao et al., 2022), MultiArith (multi-step sequential reasoning) (Roy and Roth, 2015), SVAMP (structural and textual variations) (Patel et al., 2021), and ASDiv (diverse problem types and linguistic patterns) (Xu et al., 2025), and MMLU (Hendrycks et al., 2021) and ARC-C (Clark et al., 2018). Full dataset details are provided in the Appendix B.

4.2 Baselines

We evaluate **Pre**^{2} against different representative baselines covering different reasoning paradigms: **(1) Zero-shot Pretrained**, which directly evaluates a pretrained model without fine-tuning; **(2) Few-shot In-Context Learning**, where three demonstration examples are provided as prompts without parameter updates; **(3) SFT with Explicit CoT**, which fine-tunes the model to generate the full query–reasoning–answer sequence; **(4) SFT with Frozen Embeddings**, identical to (3) but with the embedding layer fixed to assess the role of input representations; and **(5) Pause Token Baseline** (Goyal et al., 2023), which inserts fixed special tokens around the reasoning sequence. **(6) State-of-the-art (SOTA) latent CoT methods, which serve as representative architectural baselines**.

4.3 Evaluation Metrics

All evaluations are under multi-sampling evaluation (temperature = 0.1). We evaluate model performance using three metrics. **(1) Accuracy (Acc.)** measures answer correctness, defined as the percentage of correctly predicted answers. **(2) Generated Sequence Length** quantifies reasoning compactness by computing the average number of tokens in the generated reasoning chains across all examples, where shorter sequences indicate higher efficiency. **(3) Inference Throughput** evaluates efficiency by measuring the generation speed in tokens per second. Higher throughput indicates better inference efficiency.

4.4 Research Questions

Our experiments aim to answer the following questions: (1) Compared to traditional decoder-based models, what magnitude of improvement can our architecture achieve on downstream reasoning tasks? (2) More specifically, do pre-thinking modeling and pre-answering modeling enable the

Model	Method	*In-domain*	*Out-of-domain*				**Overall**
		GSM8K	**GSMHard**	**MultiArith**	**ASDiv-AUG**	**SVAMP**	**Avg**
Qwen2.5-0.5B	Zero-shot	43.06	12.15	82.78	77.55	46.33	52.37
	Few-shot	41.60	10.99	81.83	78.13	45.67	51.64
	Baseline	51.93	12.63	92.80	82.23	52.60	58.44
	Freeze-Baseline	51.93	12.61	92.63	82.19	52.13	58.30
	Pause Token	52.22	12.64	93.39	82.58	52.73	58.71
	Pre^2-1	**53.73**	**12.78**	**94.33**	**83.35**	**53.47**	**59.53**
Qwen2.5-1.5B	Zero-shot	46.90	12.40	90.00	81.31	51.33	56.38
	Few-shot	46.62	13.16	88.50	81.70	49.67	55.93
	Baseline	54.61	13.52	94.66	83.28	57.86	60.79
	Freeze-Baseline	54.45	13.39	94.53	83.27	57.80	60.69
	Pause Token	54.61	13.57	94.74	83.31	57.46	60.74
	Pre^2-1	**55.54**	**13.92**	**95.33**	**83.75**	**59.13**	**61.53**
Qwen2.5-3B	Baseline	68.41	24.76	97.7	88.27	74.33	70.71
	Freeze-Baseline	68.26	24.71	97.50	88.21	74.01	70.54
	Pause Token	68.61	25.12	97.72	88.47	75.34	71.05
	Pre^2-1	**70.13**	**26.09**	**98.66**	**89.53**	**76.33**	**72.15**
Qwen2.5-7B	Baseline	75.28	28.91	98.38	89.56	77.55	73.94
	Freeze-Baseline	75.13	28.83	98.11	89.49	77.33	73.78
	Pause Token	75.58	29.56	98.33	89.62	77.11	74.04
	Pre^2-1	**76.49**	**29.95**	**98.89**	**90.27**	**78.33**	**74.79**
Llama3.2-1B	Baseline	52.91	12.05	95.00	79.19	52.66	58.36
	Freeze-Baseline	52.53	11.75	94.17	79.09	51.00	57.71
	Pause Token	53.14	12.43	94.83	79.28	53.33	58.60
	Pre^2-1	**54.43**	**12.66**	**95.17**	**79.67**	**54.33**	**59.25**

Table 1: Performance comparison across different model scales and model series on in-domain and out-of-domain benchmarks. Statistical significance is evaluated using a paired t-test ($p < 0.05$) across 5 different training seeds. Boldfaced scores indicate that our model achieves a statistically significant improvement over the baseline.

model to predict reasoning tokens and answer tokens with higher accuracy? (3) Does scaling up the pre-thinking module and pre-answering module along the depth dimension yield consistent performance gains? (4) Similarly, does scaling up the pre-thinking scope along the width dimension enhance the model's reasoning capabilities? (5) Compared to the baseline model, we introduce two prediction modules and three internally initialized embeddings; relative to the baseline size, we add only a modest number of parameters (**with parameter increase proportions of 0.0498, 0.0478, and 0.0645 for the 0.5B, 3B, and 7B models**), and our inference mixes autoregressive decoding with latent-space prediction. Will these extra parameters and the new inference paradigm materially impact inference efficiency? We show the experimental results for these questions in Sec. 5 respectively. Refer to the appendix A for implementation details.

5 Results and Discussions

5.1 Main Results

Our model consistently outperforms baselines across in-domain and out-of-domain benchmarks. Table 1 presents a comprehensive comparison of the Qwen2.5 series across various model sizes (0.5B, 1.5B, 3B, and 7B) (Yang et al., 2025) and Llama3.2-1B (Grattafiori et al., 2024) across various reasoning benchmarks.

Several key observations emerge from these results: **(1) Consistent superiority of Pre^2-1:** Our proposed Pre^2-1 (The notation 1 denotes the use of a one-layer Transformer for modeling the pre-thinking and pre-answering modules, with the pre-thinking scope fixed at 1.) method consistently outperforms all baseline approaches across both model scales. **(2) Strong generalization to out-of-domain tasks:** Notably, Pre^2-1 demonstrates robust performance not only on the in-domain GSM8K benchmark but also across all out-of-domain datasets. **(3) Comparison with Pause Token baseline:** While the Pause Token method shows marginal improvements over the standard Baseline, it is consistently outperformed by our Pre^2-1 approach. This suggests that our pre-thinking and pre-answering mechanisms provide more effective modeling of the reasoning process than simple pause token insertion. Although both

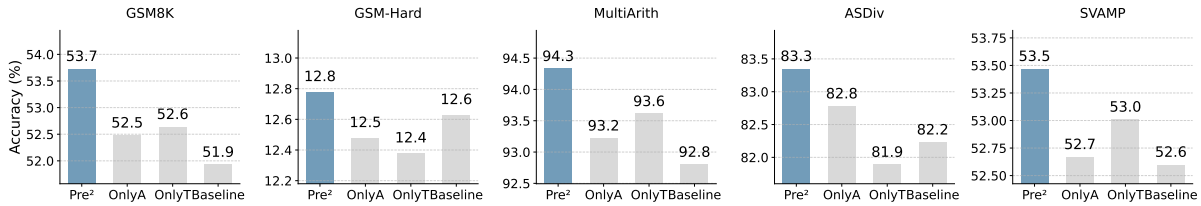

Figure 2: Ablation study: Pre-thinking module and Pre-answering module. OnlyA means we only adopt Pre-answering module and OnlyT means we only adopt Pre-thinking module.

Model	**ARC-C**	**MMLU**
Qwen-0.5B Zero-shot	29.01	47.37
Qwen-0.5B Baseline	29.12	47.38
Qwen-0.5B Freeze-baseline	29.15	47.37
Pause Token	29.37	47.41
Pre^2-1	**29.90**	**47.60**

Table 2: General reasoning performance comparison across different model variants.

Model	**GSM8K**	**GSM_H**	**Multi**	**ASDiv**	**SVAMP**	**Avg.**
Pre^2-1	53.73	12.78	94.33	83.35	53.47	59.53
Pre^2-3	**54.01**	**12.91**	**95.11**	**83.82**	**55.11**	**60.19**
Pre^2-5	53.82	12.78	94.61	83.65	54.44	59.86
Pre^2-7	53.90	12.81	94.44	83.46	53.66	59.65

Table 3: Ablation study on the Depth Dimension.

the pause token method and our approach perform equivalent scaling along the sequence width dimension, we hypothesize that this advantage arises because pause tokens employ static embeddings, whereas our pre-thinking and pre-answering embeddings are learnable and dynamic. Our novel training approach further enables these embeddings to encapsulate rich information about future token sequences, providing stronger guidance for reasoning. **(4) Scaling behavior:** The results indicate that our architectural improvements are complementary to model scaling, with benefits observed across from 0.5B to 7B parameter scales, demonstrating the generalizability of our approach. **(5) General reasoning:** We evaluate zero-shot performance on two out-of-domain general reasoning benchmarks, and the results (Table 2) show that our models achieve consistent improvements. **(6) Comparing with latent COT:** We outperform four state-of-the-art latent CoT methods (See Appendix C).

5.2 Impact of Pre-thinking and Pre-answering on Performance

As shown in Figure 2, which presents an ablation study comparing our model against the baseline and

Base Model	Ours	Size-matched Model	↑ Gain
Qwen-0.5B	53.73	52.01	**+1.72**
Qwen-1.5B	55.54	54.60	**+0.94**
Qwen-3B	70.13	68.51	**+1.62**
Qwen-7B	76.49	75.33	**+1.16**
Average	63.97	62.61	**+1.36**

Table 4: We compare our models with a param-matched transformer on GSM8K. Our method consistently improves performance within the same model-size budget.

two single-component variants, revealing the individual and synergistic contributions of our architectural components. Pre^2 achieves 59.532% average accuracy, outperforming the baseline by 1.1 points. Critically, this exceeds both OnlyT (pre-thinking only) and OnlyA (pre-answering only), demonstrating that the combined gain is substantially larger than either component alone, confirming their complementary nature. Besides, both ablated variants independently improve over the baseline.

5.3 Exploring the Depth Scaling of Pre-thinking and Pre-answering Modules

Table 3 investigates the impact of scaling the pre-thinking and pre-answering modules along the depth dimension by varying the number of Transformer layers. As the number of layers increases, performance first improves and then declines or converges. We find that the optimal depth is 3, since modeling the latent space does not require a large number of parameters. Excessive computation and parameterization may lead to overfitting.

5.4 Width Scaling of Pre-thinking Scope

Figure 3 examines the impact of scaling the pre-thinking scope(we fix the depth at 1 layer) from 1 to 5 tokens, which means we will expand the width of pre-thinking scope during training and inference. We fix the pre-answering scope at 1, as all benchmark answers are sufficiently short to be compressed into a single token representa-

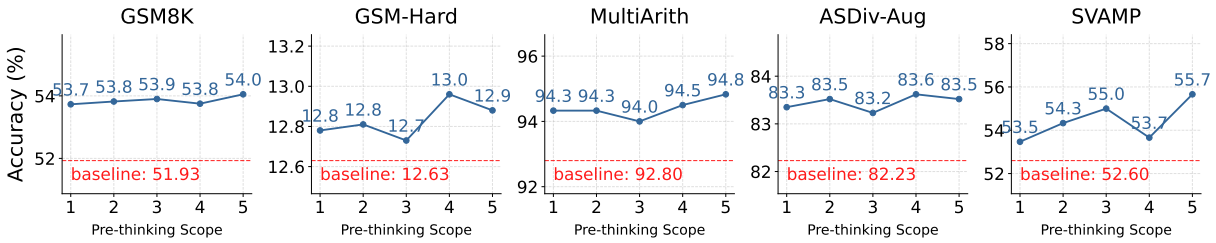

Figure 3: Ablation study on the Width Dimension (Pre-thinking Scope).

Model	GSM8K		GSMHard		MultiArith		SVAMP		ASDiv-Aug	
	Tok	T/s	Tok	T/s	Tok	T/s	Tok	T/s	Tok	T/s
Baseline	47.6	76.7 (1.00)	78.9	76.6 (0.36)	23.6	77.2 (1.31)	25.9	76.5 (1.07)	15.6	75.9 (1.45)
Pause Token	44.5	76.8 (0.96)	77.2	76.5 (0.78)	24.6	77.1 (0.55)	24.7	76.6 (1.00)	15.6	75.1 (1.52)
Pre^2-0.5b-1-1	46.1	76.9 (0.49)	78.6	76.5 (0.85)	24.6	76.3 (1.61)	24.4	76.3 (1.57)	15.6	75.4 (2.36)
Pre^2-0.5b-3-1	45.1	75.0 (1.55)	71.3	75.8 (0.81)	24.6	75.3 (1.99)	23.3	75.7 (0.90)	15.6	77.0 (0.92)
Pre^2-0.5b-5-1	44.4	74.5 (0.40)	74.9	74.4 (0.43)	24.6	74.6 (0.99)	28.5	74.3 (1.93)	15.6	75.3 (0.87)
Pre^2-0.5b-7-1	46.2	74.1 (1.59)	72.3	73.6 (1.37)	24.6	74.5 (1.29)	28.5	73.8 (2.19)	15.6	73.1 (0.88)
Pre^2-0.5b-1-2	45.0	75.8 (1.31)	77.8	76.1 (1.24)	24.6	75.7 (1.16)	26.3	76.0 (1.04)	15.6	74.2 (1.68)
Pre^2-0.5b-1-3	45.3	75.3 (1.46)	75.5	76.5 (0.88)	24.6	75.6 (1.38)	25.8	75.6 (1.23)	16.4	75.1 (0.79)
Pre^2-0.5b-1-4	45.3	75.1 (1.63)	71.0	75.2 (0.76)	24.6	75.3 (0.41)	26.4	75.4 (1.38)	15.6	75.3 (2.04)
Pre^2-0.5b-1-5	49.0	74.6 (1.04)	73.8	74.5 (0.71)	24.6	75.1 (1.31)	25.7	75.6 (0.95)	15.6	73.8 (1.69)

Table 5: Inference efficiency analysis comparing **Pre**^{2} variants against the baseline and Pause Token.

tion. Generally, expanding the pre-thinking scope demonstrates progressive performance improvements. This trend suggests that allocating wider latent space for reasoning compression enhances the model's capacity to capture complex and richer thought processes. These findings indicate that width-wise scaling of the pre-thinking module provides complementary benefits to depth scaling.

5.5 Ablation on Params-matching

As shown in Table 4, we demonstrate that performance gains stem from architectural and inference paradigm innovations rather than increased parameter count, verified by a size-matched baseline constructed by scaling the original LLM's layer depth.

5.6 Model Efficiency

Table 5 presents a comprehensive evaluation of inference throughput (tokens/second) across all model variants and benchmarks, demonstrating that our architecture maintains comparable throughput to the baseline without significant degradation.

5.7 Additional Analyses

(1) Advanced Architectural Designs: We achieve further performance gains by employing more complex pooling designs and latent-target constructions. See Table 6. **(2) Scaling Both Dimensions:** We demonstrate that simultaneously scaling both model width and depth yields additional performance improvements. See Table 12 in Appendix D. **(3) Efficiency and Parameters:** We provide a detailed breakdown of model parameters for both the baseline and our method. We report pre-filling overhead metrics to validate end-to-end efficiency. See Table 13 and Table 14 in Appendix D.

Model	GSM8K	GSM_H	Multi	ASDiv	SVAMP	Avg.
Baseline	51.93	12.63	92.80	82.23	52.60	58.44
Max	51.07	12.33	90.17	82.17	52.33	57.61
Min	50.72	11.90	89.50	82.27	51.00	57.08
Mean	53.73	12.78	94.33	83.35	53.47	59.53
Concat	**53.96**	**13.01**	**94.33**	**83.33**	**53.67**	**59.66**

Table 6: Pooling designs and latent-target construction. Concat[mean;max] achieves best performance.

6 Conclusion

We presented a phase-aware reasoning framework that decouples thinking and answering in large language models by combining guided latent representations with explicit chain-of-thought supervision. By independently modeling pre-thinking and pre-answering phases, our approach mitigates the limitations of autoregressive reasoning while preserving pretrained reasoning capabilities. Experiments across in-domain and out-of-domain benchmarks show consistent performance improvements

without inference overhead. This work highlights the importance of structured reasoning phases as a promising direction for building more efficient and robust reasoning models.

Limitations

Despite its effectiveness, our approach has several limitations. First, due to resource constraints, we did not conduct experiments on very large-scale models, such as those with more than 13B parameters. Second, our method still relies on explicit reasoning supervision during training, limiting its applicability to domains where high-quality reasoning annotations are available. We would like to emphasize that most results are come from mathematical benchmarks and another 2 eval results are from general reasoning benchmarks (mmlu and arc-c). And we think more general reasoning benchmarks should be included in the evaluation.

Acknowledgements

This work was supported by Alibaba Group through Alibaba Innovative Research Program, the Core Facility Platform of Computer Science and Communication, SIST, and the HPC platform of ShanghaiTech University.

Ethical Considerations

Our use of publicly available datasets and developed models in this research raises no ethical concerns, and we confirm compliance with the ACL Code of Ethics.

References

Lightning AI. 2023. Litgpt. https://github.com/Lightning-AI/litgpt.

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, and 30 others. 2024. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM.

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

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

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *Preprint*, arXiv:1803.05457.

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

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

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. 2023. Think before you speak: Training language models with pause tokens. *arXiv preprint arXiv:2310.02226*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

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*.

Yinhan He, Wendy Zheng, Yaochen Zhu, Zaiyi Zheng, Lin Su, Sriram Vasudevan, Qi Guo, Liangjie Hong, and Jundong Li. 2025. Semcot: Accelerating chain-of-thought reasoning through semantically-aligned implicit tokens. *arXiv preprint arXiv:2510.24940*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Preprint*, arXiv:2009.03300.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Andrew Lampinen, Ishita Dasgupta, Stephanie Chan, Kory Mathewson, Mh Tessler, Antonia Creswell, James McClelland, Jane Wang, and Felix Hill. 2022. Can language models learn from explanations in context? In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 537–563, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Xinglin Wang, Bin Sun, Heda Wang, and Kan Li. 2024. Escape sky-high cost: Early-stopping self-consistency for multi-step reasoning. *arXiv preprint arXiv:2401.10480*.

Jingyu Liu and Ce Zhang. 2025. Hamburger: Accelerating llm inference via token smashing. *arXiv preprint arXiv:2505.20438*.

Yue Liu, Jiaying Wu, Yufei He, Ruihan Gong, Jun Xia, Liang Li, Hongcheng Gao, Hongyu Chen, Baolong Bi, Jiaheng Zhang, Zhiqi Huang, Bryan Hooi, Stan Z. Li, and Keqin Li. 2025. Efficient inference for large reasoning models: A survey. *arXiv preprint arXiv:2503.23077*.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20275–20321, Suzhou, China. Association for Computational Linguistics.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. 2021. Show your work: Scratchpads for intermediate computation with language models. *Preprint*, arXiv:2112.00114.

OpenAI. 2024. Gpt-4o. Large language model.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1743–1752, Lisbon, Portugal. Association for Computational Linguistics.

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. *arXiv preprint arXiv:2502.21074*.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. 2023. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36:74952–74965.

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

Yiming Wang, Pei Zhang, Siyuan Huang, Baosong Yang, Zhuosheng Zhang, Fei Huang, and Rui Wang. 2025. Sampling-efficient test-time scaling: Self-estimating the best-of-n sampling in early decoding. *arXiv preprint arXiv:2503.01422*.

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

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

Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. 2025. Tokenskip: Controllable chain-of-thought compression in llms. *arXiv preprint arXiv:2502.12067*.

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, Vienna, Austria. 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 23 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv: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. *Advances in neural information processing systems*, 36:11809–11822.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. 2024. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2022. Automatic chain of thought prompting in large language models. In *The eleventh international conference on learning representations*.

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

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew Smith, Xingwei Qu, Mude Hui, Bohong Wu, and 14 others. 2025. Scaling latent reasoning via looped language models. *Preprint*, arXiv:2510.25741.

A Experiment Details

We provide the full hyperparameter settings, training procedures, and additional analysis used in our experiments.

A.1 Implementation Details

Our framework, **Pre**^{2}, was implemented using PyTorch(Ansel et al., 2024) and the Litgpt framework(AI, 2023), which both are open-source frameworks with permissive licenses, and our use complies with their licensing terms. We selected the **Qwen2.5** series (Yang et al., 2025) (0.5B, 1.5B, 3B and 7B) and **Llama3.2-1B**(Grattafiori et al., 2024) as our backbone models due to their strong mathematical reasoning capabilities.

Module Initialization As described in Section 3, we freeze the original embedding layer of the backbone model to maintain a stable semantic space. The pre-thinking (f_{think}) and pre-answering (f_{ans}) modules are initialized as lightweight Transformer decoder blocks. Specifically, their hidden size and number of attention heads align with the backbone model to ensure compatibility with the frozen embeddings. For the latent vectors $\mathbf{z}_{\text{think}}^{(0)}$ and $\mathbf{z}_{\text{ans}}^{(0)}$, we use a Gaussian initialization with mean 0 and standard deviation 0.02.

Training Setup All models were trained on a cluster of 8 NVIDIA A100 (80GB) GPUs using the Distributed Data Parallel (DDP) strategy. The training process utilized the AdamW optimizer. To stabilize the learning of the newly introduced latent modules alongside the backbone, we applied a linear warmup strategy followed by a cosine decay scheduler. For the baselines (Freeze-Baseline and Pause Token), we strictly followed the same training configurations (batch size, learning rate, and scheduler) to ensure a fair comparison. All evaluations are under multi-sampling evaluation (temperature = 0.1).

A.2 Hyperparameters

Table 7 lists the detailed hyperparameters used for the main experiments. We determined the loss weights λ_{think} and λ_{ans} through a preliminary grid search on a hold-out validation set from GSM8K-Aug. We found that setting the weights to 0.15 provided a balanced gradient flow between the alignment objectives ($\mathcal{L}_{\text{think}}, \mathcal{L}_{\text{ans}}$) and the autoregressive prediction objective ($\mathcal{L}_{\text{LM}}$).

B Dataset Details

In this section, we provide detailed specifications of the datasets used in our experiments. We first present the statistical overview and then detail the specific characteristics and examples for each benchmark.

B.1 Dataset Overview and Metadata

Table 8 summarizes the general metadata for each dataset. All selected datasets focus on the *Math* domain and require *Arithmetic* reasoning capabilities. The answer types are consistently numerical values.

B.2 Dataset Statistics

Table 9 presents the statistical details regarding the length of inputs and reasoning paths. We report the average length of questions (tokens), the average length of the Chain-of-Thought (CoT) rationale, and the average number of reasoning steps.

B.3 Benchmark Detail

In this section, we describe the specific characteristics of each evaluation benchmark and the training data.

GSM8K-Aug (Deng et al., 2024) GSM8K-Aug is an augmented dataset derived from GSM8K (Cobbe et al., 2021). It expands the original 8.5k training problems to roughly 385k examples through paraphrasing, numerical resampling, and

Hyperparameter	*0.5b*										*1.5b*		
	1-1	**1-2**	**1-3**	**1-4**	**1-5**	**3-1**	**5-1**	**7-1**	pt	sft	**1-1**	pt	sft
Pre-thinking Scope	1	2	3	4	5	1	1	1	1	-	1	1	-
Thinking Layers	1	1	1	1	1	3	5	7	-	-	1	-	-
Answering Layers	1	1	1	1	1	3	5	7	-	-	1	-	-
Epochs	50										11	-	-
Micro Batch Size	32										16	-	-
Max Gradient Norm	1.0												
Optimizer	AdamW												
Learning Rate	2.0e-4												
Min LR	6.0e-5												
LR Warmup Steps	2000												
Precision	bf16-mixed												
Weight Decay	0.01												
Global Batch Size	256												
β_1	0.9												
β_2	0.999												
$\lambda_{\textbf{think}}$	0.15												
$\lambda_{\textbf{ans}}$	0.15												

Table 7: Hyperparameters for all model variants. (For the format 1-1, the first 1 means we use 1-layer for Pre-thinking module and Pre-answering module and the latter 1 means Pre-thinking scope.)

Dataset	**Properties**			**Size**		**License**
	Domain	**Reasoning**	**Ans. Type**	**Train**	**Test**	
GSM8K-AUG	Math	Arithmetic	Number	385,620	1,319	MIT
GSM-Hard	Math	Arithmetic	Number	1,319	-	MIT
MultiArith	Math	Arithmetic	Number	420	180	CC BY 4.0
SVAMP	Math	Arithmetic	Number	700	300	MIT
ASDiv-AUG	Math	Arithmetic	Number	4,183	1,038	CC BY 4.0

Table 8: Overview of the datasets. We categorize the datasets by their domain properties and sample sizes.

Dataset	Avg. Q Len.	Avg. CoT Len.	Avg. Steps
GSM8K-AUG	61.3	36.2	3.26
GSM-Hard	65.8	154.2	-
MultiArith	41.3	-	-
SVAMP	41.5	15.0	1.28
ASDiv-AUG	36.6	12.3	1.23

Table 9: Detailed statistics of the **test split** used in our experiments. Lengths are calculated based on token count, and steps represent the reasoning depth of the gold solutions.

synthetic generation using GPT-4. The dataset exhibits a distribution of reasoning steps where the majority of problems require two to four steps, accompanied by a long tail of instances requiring six or more steps. This balance of common and complex instances makes the dataset suitable for training models to generalize across various reasoning difficulty levels.

GSM-Hard (Gao et al., 2022) GSM-Hard is a challenging variant of the GSM8K dataset constructed to test the ability of models to cope with harder numerical values. It retains the same problem statements as the original GSM8K but replaces many of the numbers with larger and less common numerical values, thereby increasing the difficulty of superficial arithmetic computation and reasoning. The dataset consists of approximately 1,319 examples.

MultiArith (Roy and Roth, 2015) MultiArith is a dataset of multi-step arithmetic word problems designed to challenge systems in correctly sequencing multiple operations. It contains 600 problems collected from educational sources, each solvable by a single equation involving two or more of the four basic operations (addition, subtraction, multiplication, division). The problems require reasoning over multiple sentences to extract and combine numeric quantities, understand implied operations, and compute the result. It is widely used as a benchmark for evaluating arithmetic reasoning generalization, particularly for handling compositional and sequential numerical reasoning.

SVAMP (Patel et al., 2021) SVAMP (Simple Variations on Arithmetic Math Word Problems) is a benchmark dataset designed to test the robustness of math word problem solvers to superficial changes. It contains 1,000 elementary-level arithmetic word problems (grade 4 and below), each involving a single unknown and solvable by an arithmetic expression with no more than two operators. The problems are generated through controlled variations in wording, structure, and number values applied to existing datasets (such as MAWPS and ASDiv-A) to reduce artifacts and superficial cues.

ASDiv-AUG (Xu et al., 2025) ASDiv (Academia Sinica Diverse MWP Dataset) is a diverse corpus designed to evaluate the capability of solvers across a wide range of language patterns and problem types. The original dataset contains approximately 2,305 math word problems, each annotated with its problem type and grade level to indicate difficulty. ASDiv-AUG serves as an augmented or hardened version of this benchmark, typically constructed by filtering for higher-complexity problems or generating challenging variations to demand stronger mathematical reasoning capabilities from the models.

C Comparison with Latent COT Methods

To further validate the effectiveness of **Pre**^{2}, we compare our method against several representative latent Chain-of-Thought baselines, including Coconut (Hao et al., 2024), iCoT (Deng et al., 2024), CODI (Shen et al., 2025), and two SimCoT variants (Wei et al., 2025), all built on the LLaMA-3.2-1B backbone. As shown in Table 11, our **Pre**^{2} consistently surpasses all latent CoT competitors across the five reasoning benchmarks, indicating that the proposed hybrid latent–token paradigm preserves reasoning fidelity better than purely latent alternatives.

D Additional Analyses

This section provides supplementary experimental results that complement the findings reported in the main paper. Together, these analyses confirm that **Pre**^{2} delivers consistent gains with negligible parameter and latency overhead.

D.1 Jointly scaling

Specifically, we analyze the benefit of jointly scaling both the width (pre-thinking scope) and depth (number of Transformer layers) dimensions (Table 12),

D.2 Inference latency

We report fine-grained inference latency in terms of Time-to-First-Token and per-token decoding cost (Table 13),

Dataset	Example Question	Rationale / Steps	Answer
GSM8K-Aug	Out of 600 employees in a company, 30% got promoted while 10% received bonus. How many employees did not get either a promotion or a bonus?	["«600*30/100=180»", "«600*10/100=60»", "«180+60=240»", "«600-240=360»"]	360
GSM-Hard	Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with 4933828. She sells the remainder at the farmers' market daily for $2 per fresh duck egg. How much in dollars does she make every day at the farmers' market?	def solution(): """Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for $2 per fresh duck egg. How much in dollars does she make every day at the farmers' market?""" eggs_per_day = 16 eggs_eaten = 3 eggs_baked = 4933828 eggs_sold = eggs_per_day - eggs_eaten - eggs_baked price_per_egg = 2 money_made = eggs_sold * price_per_egg result = money_made return result	-9,867,630
MultiArith	There are 64 students trying out for the school's trivia teams. If 36 of them didn't get picked for the team and the rest were put into 4 groups, how many students would be in each group?	-	7
SVAMP	There are 87 oranges and 290 bananas in Philip's collection. If the bananas are organized into 2 groups and oranges are organized into 93 groups How big is each group of bananas?	(290.0 / 2.0)	145
ASDiv-AUG	gino has 64 popsicle sticks . i have 100 popsicle sticks . what is the sum of our popsicle sticks ?	«64+100=164»	164

Table 10: Examples from the evaluation and training datasets. Each entry shows a representative question, the reasoning steps (Rationale) required to solve it, and the final numerical answer.

Model	GSM8K	GSM-Hard	MultiArith	ASDiv	SVAMP
llama-3.2-1b-baseline	52.91	12.05	95.00	79.19	52.66
llama-3.2-1b-freeze-baseline	52.53	11.75	94.17	79.09	51.00
llama-3.2-1b-pause token	53.14	12.43	94.83	79.28	53.33
llama-3.2-1b-1 (ours)	**54.43**	**12.66**	**95.17**	**79.67**	**54.33**
llama-3.2-1b-coconut	45.30	9.90	90.10	–	48.80
llama-3.2-1b-icot	19.00	4.40	39.00	–	40.90
llama-3.2-1b-codi	45.56	10.46	91.83	77.36	50.32
llama-3.2-1b-simcot-coconut	49.27	10.61	92.16	78.13	51.01
llama-3.2-1b-simcot-codi	50.03	11.06	92.67	79.09	51.67

Table 11: Performance comparison of LLaMA-3.2-1B variants on five benchmarks.

D.3 Parameter overhead

We summarize the parameter overhead introduced by our additional modules relative to the original backbones (Table 14).

D.4 Larger thinking scope for larger model

We repeated this ablation study on larger models and variants with a larger thinking scope (Table 15), and found that:

- OnlyA and OnlyT are generally better than the baseline when we scaling the model size.
- Pre-thinking brings larger improvements than pre-answering.

We think that the new two experiments, together with the previous experiments on pre-1-1-0.5b, brining the following points:

- Learning in a limited guided latent space is challenging for smaller models.
- In order to achieve the potential of the guided latent space, smaller models should use a broader thinking scope during both training and inference.

E Inference Complexity Analysis

We provide a detailed analysis of the inference complexity of Pre^2 and compare it with standard autoregressive decoding. We distinguish the one-time cost of latent prediction from the additional attention cost introduced by the latent guidance tokens.

E.1 Notation

Let M denote the query length, N_c the number of generated chain-of-thought (CoT) tokens, and N_a the number of generated answer tokens. We define

$$N = N_c + N_a. \tag{5}$$

Let L and d denote the number of layers and hidden dimension of the backbone language model, respectively. The latent predictor contains two parallel Transformer branches, each with $l = 1$ layer.

For a Transformer layer operating on a sequence of length S, we use the standard computational cost model

$$\mathcal{C}_{\text{layer}}(S) = \Theta\big(Sd^2 + S^2d\big)\,, \tag{6}$$

where the first term accounts for linear projections and feed-forward transformations, while the second term accounts for self-attention.

During autoregressive decoding with KV caching, only the newly generated token is processed by the projection and feed-forward blocks. If the current attention context contains S tokens, the cost of one decoding step is therefore

$$\mathcal{C}_{\text{step}}(S) = \Theta\big(d^2 + Sd\big) \tag{7}$$

per layer, and hence

$$\mathcal{C}^{(L)}_{\text{step}}(S) = \Theta\big(Ld^2 + LSd\big)\,. \tag{8}$$

We focus on the dominant Transformer computation and omit terms that are common to both methods, such as the final vocabulary projection, as well as constant factors associated with the number of attention heads.

E.2 Baseline Autoregressive Inference

The baseline inference process consists of query prefilling followed by autoregressive generation.

Query prefilling. The query contains M tokens and is processed by all L backbone layers without KV caching. By Eq. (6), its cost is

$$\mathcal{C}_{\text{prefill}} = \Theta\big(M^2Ld + MLd^2\big)\,. \tag{9}$$

Autoregressive decoding. At the t-th generation step, the newly generated token attends to the M query tokens and the preceding $t-1$ generated tokens. Consequently, its attention context has length

$$S_t^{\text{base}} = M + t - 1. \tag{10}$$

Using Eq. (8), the total decoding cost is

$$\begin{aligned}\mathcal{C}^{\text{base}}_{\text{dec}} &= \Theta\left(NLd^2 + Ld\sum_{t=1}^{N}(M+t-1)\right)\\ &= \Theta\left(NLd^2 + Ld\left[NM + \frac{N(N-1)}{2}\right]\right).\end{aligned} \tag{11}$$

Therefore, the total baseline inference cost is

$$\mathcal{C}_{\text{base}} = \mathcal{C}_{\text{prefill}} + \mathcal{C}^{\text{base}}_{\text{dec}}. \tag{12}$$

E.3 Inference Complexity of Pre^2

The Pre^2 inference pipeline consists of five stages: (i) query prefilling, (ii) parallel latent prediction, (iii) guided CoT decoding, (iv) state transition, and (v) guided answer decoding.

Method	GSM8K	GSM-Hard	MultiArith	ASDiv-Aug	SVAMP
Baseline	51.93	12.63	92.80	82.23	52.60
Pre^2-1-1	53.73	12.78	94.33	83.35	53.47
Pre^2-3-1	54.01	12.91	95.11	83.82	55.11
Pre^2-3-3	**54.21**	**13.12**	**95.33**	**83.91**	**55.33**

Table 12: Performance comparison of different Pre^2 configurations across five reasoning benchmarks. Best results are in **bold**.

Metric	Baseline	Pre^2	Abs.	Rela.
Prefilling Latency (TTFT)	117.519	118.031	+0.512	+0.43%
Model VRAM	1.174	1.232	+0.058	+4.99%
Generation Length	47.60	46.10	−1.50	−3.15%
End-to-End Latency	721.480	736.871	+15.391	+2.09%
Serving Throughput	76.81	75.35	−1.46	−1.90%

Table 13: Inference efficiency comparison between the baseline and Pre^2. "Abs." denotes the absolute difference, while "Rela." denotes the relative change.

Model	Original Params	Total Params	Trainable Params	Non-Trainable Params	Added Params	Increase Ratio
Qwen2.5-0.5B	630,167,424	661,602,304	525,467,648	136,134,656	31,434,880	**0.0498**
Qwen2.5-3B	3,397,103,616	3,559,656,448	3,248,491,520	311,164,928	162,552,832	**0.0478**
Qwen2.5-7B	7,615,616,512	8,107,440,128	7,562,442,752	544,997,376	491,823,616	**0.0645**

Table 14: Parameter statistics for different Qwen2.5 model variants. The **Increase Ratio** indicates the proportion of added parameters relative to the original model size.

Method	GSM8K	GSM-Hard	MultiArith	ASDiv-Aug	SVAMP
Qwen-3B					
Baseline	68.26	24.71	97.50	88.21	74.01
1	**70.13**	**26.09**	**98.66**	**89.53**	**76.33**
1-Only$_t$	68.96	24.84	98.11	88.34	74.88
1-Only$_a$	68.63	24.99	97.94	88.47	74.66
Qwen-0.5B					
Baseline	51.93	12.63	92.80	82.23	52.60
1-5	**53.75**	**12.96**	**94.50**	**83.62**	**53.66**
1-5-Only$_t$	52.28	12.66	93.11	82.59	52.78
1-5-Only$_a$	52.11	12.76	92.94	82.31	52.56

Table 15: Ablation study of pre-thinking module and pre-answering module on small and larger models.

E.3.1 Stage 1: Query Prefilling

The first stage is identical to baseline query prefilling. Therefore,

$$\mathcal{C}_{\text{prefill}}^{Pre^2} = \Theta\big(M^2Ld + MLd^2\big). \tag{13}$$

In particular,

$$\mathcal{C}_{\text{prefill}}^{Pre^2} = \mathcal{C}_{\text{prefill}}. \tag{14}$$

Thus, query prefilling introduces no additional computational overhead relative to the baseline.

E.3.2 Stage 2: Parallel Guided Latent Prediction

After query prefilling, the hidden states of the query are fed into two parallel one-layer Transformer predictors. Each predictor receives the query representation together with one trigger token, and therefore operates on a sequence of length $M+1$.

The computational cost of one predictor is

$$\mathcal{C}_{\text{pred}}^{(1)} = \Theta\big((M+1)d^2 + (M+1)^2d\big). \tag{15}$$

Since there are two predictor branches, their total computational work is

$$\mathcal{C}_{\text{pred}}^{\text{work}} = \Theta\big(2(M+1)d^2 + 2(M+1)^2d\big). \tag{16}$$

For the latency analysis, however, the two branches are executed in parallel. Assuming balanced branches and sufficient hardware parallelism, their idealized critical-path latency is that of a single branch:

$$\mathcal{C}_{\text{pred}}^{\text{lat}} = \Theta\big((M+1)d^2 + (M+1)^2d\big). \tag{17}$$

For $M \gg 1$, Eq. (17) becomes

$$\mathcal{C}_{\text{pred}}^{\text{lat}} = \Theta\big(Md^2 + M^2d\big). \tag{18}$$

E.3.3 Stage 3: Guided CoT Decoding

Before CoT decoding begins, the latent vector $\mathbf{z}_{cot}^*$ is inserted into the attention context. At the t-th CoT decoding step, the context therefore consists of the M query tokens, the inserted latent vector, and the $t-1$ previously generated CoT tokens. Hence,

$$S_t^c = M + 1 + t - 1 = M + t. \tag{19}$$

Using Eq. (8), the total cost of generating N_c CoT tokens is

$$\begin{aligned}\mathcal{C}_c &= \Theta\left(N_cLd^2 + Ld\sum_{t=1}^{N_c}(M+t)\right)\\ &= \Theta\left(N_cLd^2 + Ld\left[N_cM + \frac{N_c(N_c+1)}{2}\right]\right).\end{aligned} \tag{20}$$

E.3.4 Stage 4: State Transition

After the CoT sequence is completed, Pre^2 appends the end-of-thought marker $\mathbf{e}_{EOT}$ and the precomputed answer latent $\mathbf{z}_{ans}^*$ to the KV cache.

We assume that $\mathbf{z}_{ans}^*$ is already represented in the form required by the KV cache, so that this operation only involves a constant number of memory writes. Under this implementation assumption,

$$\mathcal{C}_{\text{transition}} = \Theta(1). \tag{21}$$

Importantly, no backbone forward pass is performed during this stage.

E.3.5 Stage 5: Guided Answer Decoding

At the beginning of answer decoding, the attention context contains the M query tokens, the CoT latent $\mathbf{z}_{cot}^*$, the N_c generated CoT tokens, the EOT marker, and the answer latent $\mathbf{z}_{ans}^*$.

Thus, before generating the t-th answer token, the attention context has length

$$S_t^a = M + N_c + t + 2. \tag{22}$$

The resulting answer-decoding cost is

$$\begin{aligned}\mathcal{C}_a &= \Theta\left(N_aLd^2 + Ld\sum_{t=1}^{N_a}(M+N_c+t+2)\right)\\ &= \Theta\left(N_aLd^2 + Ld\left[N_a(M+N_c+2) + \frac{N_a(N_a+1)}{2}\right]\right).\end{aligned} \tag{23}$$

E.4 Total Decoding Complexity of Pre^2

Combining Stages 3–5, the decoding cost of Pre^2 is

$$\mathcal{C}_{\text{dec}}^{Pre^2} = \mathcal{C}_c + \mathcal{C}_{\text{transition}} + \mathcal{C}_a. \tag{24}$$

Substituting Eqs. (20), (21), and (23) gives

$$\begin{aligned}\mathcal{C}_{\text{dec}}^{Pre^2} = \Theta\Bigg(NLd^2 + Ld\Bigg[&N_c(M+1) + N_a(M+N_c+2)\\ &+ \frac{N_c(N_c+1)}{2} + \frac{N_a(N_a+1)}{2}\Bigg]\Bigg).\end{aligned} \tag{25}$$

E.5 Additional Attention Cost

We next characterize the additional attention computation introduced by the latent guidance tokens.

Proposition 1. *Under the causal attention pattern described above, the additional attention-context computation of Pre^2 relative to the baseline is*

$$\Delta\mathcal{C}_{\text{attn}} = \Theta\big((N_c + 3N_a)Ld\big). \tag{26}$$

Consequently,

$$\Delta\mathcal{C}_{\text{attn}} \leq \Theta(3NLd). \tag{27}$$

Proof. For the baseline, the total number of attention-context tokens processed during decoding is

$$S_{\text{base}} = \sum_{t=1}^{N}(M+t-1) = NM + \frac{N(N-1)}{2}. \tag{28}$$

For Pre^2, the corresponding quantity is

$$\begin{aligned} S_{Pre^2} &= \sum_{t=1}^{N_c}(M+t) \\ &\quad + \sum_{t=1}^{N_a}(M+N_c+t+2) \\ &= N_cM + \frac{N_c(N_c+1)}{2} \\ &\quad + N_a(M+N_c+2) + \frac{N_a(N_a+1)}{2}. \end{aligned} \tag{29}$$

Subtracting Eq. (28) from Eq. (29), and using $N = N_c + N_a$, yields

$$\begin{aligned} S_{Pre^2} - S_{\text{base}} &= N_c + N_aN_c + 3N_a \\ &\quad + \frac{N_c^2 + N_a^2 - N^2}{2}. \end{aligned} \tag{30}$$

Since

$$N^2 = N_c^2 + 2N_cN_a + N_a^2, \tag{31}$$

we have

$$\frac{N_c^2 + N_a^2 - N^2}{2} = -N_cN_a. \tag{32}$$

Therefore,

$$\begin{aligned} S_{Pre^2} - S_{\text{base}} &= N_c + N_aN_c + 3N_a - N_cN_a \\ &= N_c + 3N_a. \end{aligned} \tag{33}$$

Each additional context position contributes $\Theta(Ld)$ attention computation. Hence,

$$\Delta\mathcal{C}_{\text{attn}} = \Theta((N_c + 3N_a)Ld)\,. \tag{34}$$

Finally, because $N_c, N_a \geq 0$,

$$N_c + 3N_a \leq 3(N_c + N_a) = 3N. \tag{35}$$

Thus,

$$\Delta\mathcal{C}_{\text{attn}} \leq \Theta(3NLd). \tag{36}$$

This result means that each CoT token attends to one additional latent token, $\mathbf{z}^*_{cot}$, resulting in N_c additional attention-context positions. Each answer token, in turn, attends to three additional positions: $\mathbf{z}^*_{cot}$, $\mathbf{e}_{EOT}$, and $\mathbf{z}^*_{ans}$. Hence, the exact increase is $N_c + 3N_a$.

E.6 One-time Latent Prediction Overhead

The latent predictor is executed only once per input query. For ideal parallel execution, Eq. (18) gives

$$\mathcal{C}^{\text{lat}}_{\text{pred}} = \Theta\big(Md^2 + M^2d\big)\,. \tag{37}$$

We define the prediction overhead ratio with respect to the baseline decoding computation as

$$R_{\text{pred}} = \frac{\mathcal{C}^{\text{lat}}_{\text{pred}}}{\mathcal{C}^{\text{base}}_{\text{dec}}}. \tag{38}$$

From Eq. (11),

$$\mathcal{C}^{\text{base}}_{\text{dec}} = \Theta\left(NLd^2 + Ld\left[NM + \frac{N(N-1)}{2}\right]\right). \tag{39}$$

Since the denominator is at least $\Theta(NLd^2)$, we obtain the upper bound

$$\begin{aligned} R_{\text{pred}} &\leq \Theta\left(\frac{Md^2 + M^2d}{NLd^2}\right) \\ &= \Theta\left(\frac{M}{NL} + \frac{M^2}{NLd}\right). \end{aligned} \tag{40}$$

When $M \ll d$, the second term is dominated by the first, giving

$$R_{\text{pred}} = \Theta\left(\frac{M}{NL}\right). \tag{41}$$

E.7 Attention-Context Overhead Ratio

Using Proposition 1, the additional attention overhead ratio is

$$R_{\text{attn}} = \frac{(N_c + 3N_a)Ld}{NLd^2 + Ld\left[NM + \frac{N(N-1)}{2}\right]}. \tag{42}$$

Cancelling the common factor Ld gives

$$R_{\text{attn}} = \frac{N_c + 3N_a}{Nd + NM + \frac{N(N-1)}{2}}. \tag{43}$$

Since the denominator is at least Nd and $N_c + 3N_a \leq 3N$, we obtain

$$R_{\text{attn}} \leq \frac{3N}{Nd} = \frac{3}{d}. \tag{44}$$

Thus, the relative overhead caused by the additional attention context decreases inversely with the hidden dimension d.

E.8 Overall Inference Overhead

We now combine the one-time latent prediction cost and the additional attention-context cost.

Theorem 1. *Assume that (i) the two latent predictors execute in parallel, (ii) $\mathbf{z}^*_{cot}$ and $\mathbf{z}^*_{ans}$ can be inserted into the KV cache without an additional backbone forward pass, and (iii) $M \ll d$. Then the idealized inference-latency overhead of Pre^2 relative to baseline autoregressive decoding satisfies*

$$R_{\text{total}} = \mathcal{O}\left(\frac{M}{NL} + \frac{3}{d}\right). \tag{45}$$

Proof. By Eq. (14), the query-prefilling costs of Pre^2 and the baseline are identical and therefore cancel when computing the additional inference cost.

The remaining additional cost consists of two components: the one-time latent prediction cost and the additional attention-context cost. By Eq. (41), the former satisfies

$$R_{\text{pred}} = \mathcal{O}\left(\frac{M}{NL}\right). \tag{46}$$

By Eq. (44), the latter satisfies

$$R_{\text{attn}} \leq \frac{3}{d}. \tag{47}$$

Therefore,

$$\begin{aligned} R_{\text{total}} &= R_{\text{pred}} + R_{\text{attn}} \\ &= \mathcal{O}\left(\frac{M}{NL} + \frac{3}{d}\right). \end{aligned} \tag{48}$$

Discussion. Theorem 1 reveals two important properties of Pre^2. First, the one-time latent prediction overhead decreases with the generation length N and backbone depth L, scaling as $\mathcal{O}(M/(NL))$ under ideal parallel execution. Second, the additional attention-context overhead is bounded by $\mathcal{O}(1/d)$, which becomes increasingly small for large-width language models. Therefore, Pre^2 introduces sublinear and highly amortized inference overhead while providing additional latent guidance for both reasoning and answer generation.

Remark on Computational Work versus Latency. The bound in Theorem 1 concerns idealized inference latency, for which the two latent-prediction branches are executed concurrently. If computational work (e.g., total FLOPs) is used instead, both predictor branches must be counted. Consequently,

$$R^{\text{work}}_{\text{pred}} = \mathcal{O}\left(\frac{2M}{NL}\right) \tag{49}$$

under $M \ll d$, while the attention-context term remains bounded by $\mathcal{O}(3/d)$.

F Ai Assistants In Research Or Writing

We used chatGPT(OpenAI, 2024) only for grammar checking and minor typo correction. No prompt injections are included in this submission.