MemoSight: Unifying Context Compression and Multi-Token Prediction for Reasoning Acceleration

Anonymous ACL submission

Abstract

While chain-of-thought (CoT) reasoning enables LLMs to solve challenging reasoning tasks, the linear growth of the KV cache leads to substantial memory and inference overhead. Existing approaches such as context compression and multi-token prediction (MTP) improve efficiency from two complementary directions by compressing historical tokens and generating future tokens in parallel. However, effectively combining them remains challenging due to their different training paradigms and architectural assumptions. In this work, we propose MemoSight (**Memo**ry-Fore**sight**-Based Reasoning), a unified framework that integrates context compression and MTP to improve inference efficiency while preserving CoT performance. MemoSight adopts a shared minimalist design based on special tokens and token-specific positional layouts for both compression and parallel prediction. Experiments on four reasoning benchmarks show that, compared to the vanilla SFT baseline, MemoSight reduces KV cache usage by up to 66% and improves inference speed by 56%, while incurring less than a 3% drop in average reasoning accuracy, yielding a better efficiency–accuracy trade-off than existing CoT compression methods.

1 Introduction

Long Chain-of-Thought (CoT) reasoning has become an important mechanism for improving the abilities of large language models (LLMs) (Zhao et al., 2023; Guo et al., 2025; Jaech et al., 2024). However, in Transformer-based architectures (Vaswani et al., 2017), the Key-Value (KV) cache grows linearly with the number of generated tokens, creating latency and memory bottlenecks that hinder the real-world deployment of long CoT reasoning (Arora and Zanette, 2025).

Context compression and multi-token prediction (MTP) address these inference bottlenecks from complementary aspects. **Context compression** (Chang et al., 2024; Zhang et al., 2025a) condenses reasoning traces into compact "memory tokens", allowing the model to represent past reasoning states with a much shorter context during inference. This reduces KV cache usage and lowers decoding cost. **Multi-token prediction (MTP)** (Gloeckle et al., 2024; Liu et al., 2024), in contrast, trains models to predict multiple future tokens at each step rather than only the next token. When combined with speculative decoding (Cai et al., 2024a; Li et al., 2024b), MTP can further speed up inference. In short, context compression reduces the cost of processing *past* reasoning traces, while MTP accelerates *future* token generation. Their combination offers the potential for further inference efficiency gains.

However, we find that a naive combination of MTP and context compression degrades performance, as shown in Section 4.3. Specifically, we follow the MTP design of DeepSeek-V3 (Liu et al., 2024) by augmenting the model with additional Transformer blocks and fine-tuning the resulting model on the same context compression data. We hypothesize that this failure arises from a mismatch between conventional MTP designs and the context compression setting. Existing MTP methods are mainly developed for large-scale pre-training and introduce substantial architectural changes with many newly added parameters. In contrast, context compression is learned during supervised fine-tuning through lightweight adaptation. As a result, conventional MTP designs are not well aligned with the SFT-based context compression objective, leading to suboptimal performance.

In this work, we integrate context compression and MTP in our proposed MemoSight (**Memo**ry-Fore**sight**-Based Reasoning) framework. MemoSight requires no modifications to LLM architectures and provides a special-token-based unified framework through the following design:

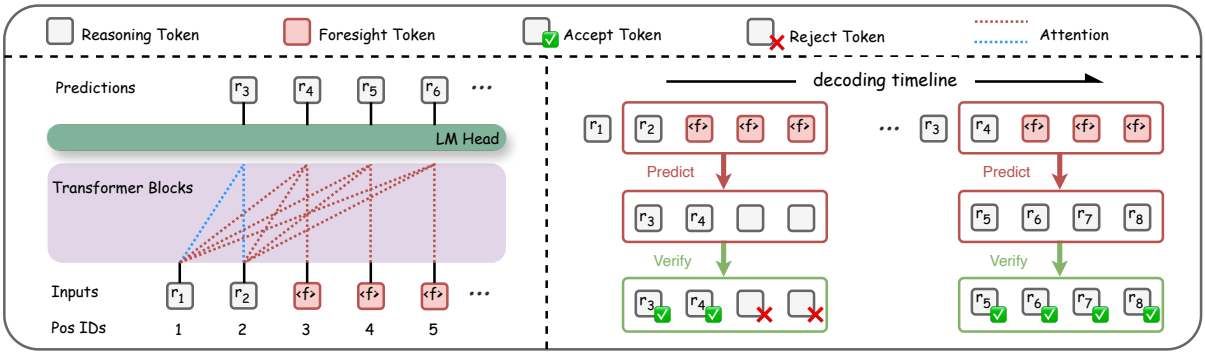

Figure 1: Illustration of **foresight-token-based acceleration** ($d = 3$). **(Left)** Foresight tokens $\langle \mathrm{f} \rangle$ are inserted after the current reasoning prefix with increasing position IDs; each foresight token attends to reasoning tokens and itself, predicting a future reasoning token through the shared LM head. **(Right)** During decoding, the model first generates multiple candidate future tokens in a single forward pass (*Predict*), then verifies them in parallel (*Verify*). Correct predictions are accepted and appended to the reasoning sequence, while incorrect predictions are rejected.

- **Special Tokens as Carriers.** Inspired by recent developments from context compression (Zhang et al., 2025a) and MTP (Gerontopoulos et al., 2025), we incorporate both mechanisms into our framework by leveraging special tokens with different roles: memory tokens condense reasoning steps into compact representations; foresight tokens trigger multi-token prediction.
- **Position-Aware Alignment.** Prior work has shown effectiveness by designing position layout for context compression (Zhao et al., 2024) and MTP (Gerontopoulos et al., 2025) in isolation. MemoSight introduces a tailored position layout for each type of special tokens, aligning memory tokens with past reasoning and foresight tokens with future predictions.

By integrating context compression and MTP within a unified framework, MemoSight combines the benefits of both methods. As we demonstrate in Section 3, evaluations on four reasoning benchmarks show that, compared to the vanilla CoT fine-tuning baseline, MemoSight reduces KV cache usage by 66% and improves inference speed by 56%, with only a 3% average accuracy drop. Compared to the context compression baseline LightThinker, MemoSight improves average accuracy by 3 points and inference speed by 27%, yielding a better efficiency–performance tradeoff.

2 MemoSight

This section presents MemoSight (Memory-Foresight-Based Reasoning), a framework that jointly models context compression and multi-token prediction (MTP) for efficient CoT reasoning. Notably, MemoSight employs unified techniques through special tokens and dedicated positional layout for both context compression and MTP, requiring no architectural modifications; such unified and minimalist design enables MemoSight to achieve benefits from both techniques through supervised fine-tuning. In this section, we first introduce the two inference accelerations (Section 2.1) MemoSight incorporates, then how we train MemoSight to enable these two accelerations: Section 2.2 describes the training sequence construction, followed by the training framework with customized attention (Section 2.3).

2.1 Iterative Inference Pipeline

Given a prompt $P = [p_1, \ldots, p_{|P|}]$, a reasoning LLM parameterized by θ autoregressively generates a reasoning sequence $R = [r_1, \ldots, r_{|R|}]$ according to $p_\theta(r_t \mid P, r_{<t})$. MemoSight modifies this decoding process by alternating between accelerated generation and context compression. To support these inference modes, MemoSight introduces two types of special tokens [1]: *foresight tokens* $\langle f \rangle$ for MTP and *memory tokens* $\langle m \rangle$ for context compression.

2.1.1 Foresight Token-Based Acceleration

Formally, MTP predicts multiple future tokens according to $p_\theta(r_{t:t+d} \mid P, r_{<t})$, where d is the MTP horizon and corresponds to the number of tokens predicted beyond the next token. Conventional methods achieve this by branching the current hidden states into d parallel (Gloeckle et al., 2024) or sequential (Liu et al., 2024) prediction heads, each responsible for predicting a different future offset.

[1] Throughout the paper, we use $\langle \cdot \rangle$ to denote special tokens introduced for modeling purposes.

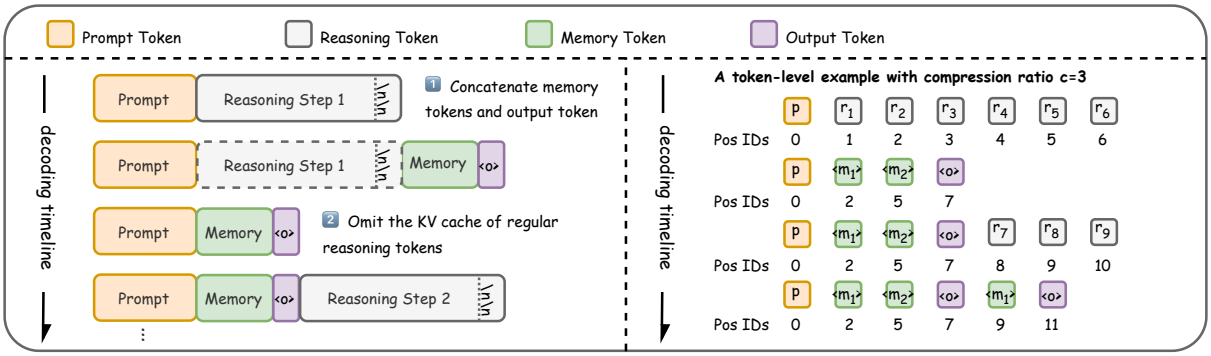

Figure 2: Illustration of **memory-token-based compression**. **(Left)** After each reasoning step, the model compresses the reasoning trace into memory tokens and discards the verbose KV cache. Future reasoning attend only to the prompt, memory tokens, and output tokens. **(Right)** A token-level example with compression ratio $c = 3$. Memory token count scales with the reasoning step length (AMA), and their position IDs are uniformly interpolated over the compressed span (UPL).

Motivated by (Gerontopoulos et al., 2025), MemoSight instead performs MTP using special tokens without modifying the model architecture. We append d foresight tokens $\langle \mathrm{f} \rangle$ after the current position and model

$$p_\theta\Big(r_{t:t+d} \;\Big|\; P, r_{<t}, \underbrace{\langle \mathrm{f} \rangle, \ldots, \langle \mathrm{f} \rangle}_{d}\Big). \tag{1}$$

As illustrated in Figure 1(left), each foresight token corresponds to a different future offset and is assigned a sequentially increasing position ID, indicating which future token it predicts (e.g., $\langle \mathrm{f} \rangle$ with position ID 3 predicts the token r_4 at position 4). Each foresight token attends only to preceding regular tokens and itself, preventing information leakage across future offsets. As a result, a single forward pass produces $d+1$ draft tokens, including the standard next-token prediction from the current token representation.

As shown in Figure 1(right), during inference, we append d foresight tokens after the current position to generate $d + 1$ draft tokens in a single forward pass (with $d = 3$ in the figure). We then perform a parallel verification pass over all draft tokens, keep the accepted tokens [2], and continue generation from the last accepted position. For example, in Figure 1(right), r_3, r_4 are accepted in the first MTP, which are used in the following predcitions.

2.1.2 Memory-Token-Based Compression

Following prior work (Zhang et al., 2025a), MemoSight partitions the reasoning sequence $R = [r_1, \ldots, r_{|R|}]$ into a sequence of reasoning steps $R_1, \ldots, R_n$ using a delimiter. [3] Once the model generates the step delimiter token, it appends both memory tokens and a dedicated output token $\langle \mathrm{o} \rangle$ after the current step R_i. The memory tokens compress R_i into compact representations, while $\langle \mathrm{o} \rangle$ enables continued generation over the compressed context. The regular reasoning tokens in R_i are then removed from the KV cache to reduce the effective context length.

We illustrate this memory token based compression process in of Figure 2 (left); on top of this, MemoSight further introduces two improvements to enhance the compression effectiveness that we illustrate in Figure 2 (right):

Adaptive Memory Allocation (AMA). Prior methods assign a fixed number of memory tokens to each reasoning step. However, reasoning steps can vary substantially in length and information density. Instead, MemoSight allocates memory tokens according to a predefined compression ratio c. For each reasoning step R_i, we compute the number of memory tokens as $l = \lceil |R_i|/c \rceil$ and append memory tokens $M_i = [m_1, \ldots, m_l]$ after R_i. AMA thus enables more flexible compression for long reasoning traces.

Uniform Position Layout (UPL). Prior methods assign consecutive new position IDs to memory tokens. After reasoning tokens are evicted, the remaining memory-token positions become sparsely distributed with large positional gaps, disrupting relative positional relationships in attention mechanisms such as RoPE (Su et al., 2024). Motivated

[2] For example, we accept the prefix of tokens, each of which coincide with argmax prediction.

[3] Following (Zhang et al., 2025a), we use "\n\n" as our delimiter.

by Zhao et al. (2024), MemoSight instead assigns memory-token position IDs by uniformly interpolating across the positional span of each reasoning step. Specifically, we divide the current reasoning step R_i into $l = \lceil |R_i|/c \rceil$ sequential groups,[4] and assign each memory token the center position of its corresponding group. This preserves a uniform positional distribution of memory tokens across the historical context. Moreover, UPL places memory tokens closer to their corresponding content in positional space, facilitating better compression.

2.1.3 MemoSight Inference

During inference, MemoSight alternates between foresight-token-based acceleration and memory-token-based compression [5]. Within each reasoning step R_i, the model performs parallel future-token prediction for accelerated decoding. Once R_i is completed, the model compresses it into memory tokens M_i and continues generation over the compressed context. These two inference modes are complementary: foresight-token acceleration reduces the number of decoding iterations through parallel generation, while memory-token-based compression controls KV cache growth during long reasoning processes. When MTP and compression are well trained, their combination is expected to improve both inference speed and memory efficiency for CoT reasoning.

2.2 Data Construction Pipeline

To train MemoSight to perform context compression and MTP, we construct training instances from standard CoT trajectories by inserting memory tokens $\langle \mathrm{m} \rangle$ and foresight tokens $\langle \mathrm{f} \rangle$ with customized position IDs.

CoT Augmentation. Given a CoT trajectory $R = [R_1, \ldots, R_n]$, where each reasoning step is represented as $R_i = [r_1^i, r_2^i, \ldots, r_{|R_i|}^i]$, we first augment each reasoning step with foresight tokens. Specifically, for each token r_t^i, we insert a foresight token $\langle \mathrm{f} \rangle$ immediately before it, yielding

$$\tilde{R}_i = [\langle \mathrm{f} \rangle, r_1^i, \langle \mathrm{f} \rangle, r_2^i, \ldots, \langle \mathrm{f} \rangle, r_{|R_i|}^i].$$

During training, each foresight token is optimized to predict the token d' steps ahead, where $d' \in \{0, \ldots, d\}$ is randomly sampled for each training sample [6]. This enables the model to learn future token prediction under different offsets.

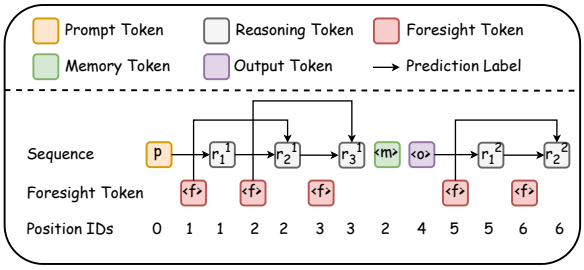

Figure 3: An example of sequence $\mathcal{S}$ with a compression ratio $c = 3$ and foresight offset $d' = 1$. The reasoning tokens (r_1^1, r_2^1, r_3^1) are compressed into the memory token $\langle \mathrm{m} \rangle$ with an interpolated position ID (i.e., 2). An output token $\langle \mathrm{o} \rangle$ then bridges the compressed context to the subsequent step. Foresight tokens $\langle \mathrm{f} \rangle$ are inserted before r_t^i with position IDs $t + d' - 1$. The solid arrows indicate the prediction labels for each token type.

For each reasoning step except the final one, we further append memory tokens $M_i = [\langle \mathrm{m}_1 \rangle, \ldots, \langle \mathrm{m}_l \rangle]$ after $\tilde{R}_i$, where $l = \lceil |R_i|/c \rceil$ is determined by the compression ratio c, followed by an output token $\langle \mathrm{o} \rangle$ marking the transition to the next reasoning step. Combined with the prompt P, the final training sequence is constructed as

$$\mathcal{S} = [P, \tilde{R}_1, M_1, \langle \mathrm{o} \rangle, \tilde{R}_2, M_2, \langle \mathrm{o} \rangle, \ldots, \tilde{R}_n].$$

Label Assignment. We define supervision targets for different token types in the sequence $\mathcal{S}$. As illustrated in Figure 3, excluding the prompt P, $\mathcal{S}$ contains four token types: reasoning, foresight, output, and memory tokens.

Reasoning tokens r_t^i follow standard next-token prediction within each reasoning step $\tilde{R}_i$. No prediction target is assigned to the last reasoning token in $\tilde{R}_i$ (i.e., the step delimiter token "\n\n").

For each foresight token $\langle \mathrm{f} \rangle$ preceding r_t^i, the prediction target is defined as $r_{t+d'}^i$. If $t+d' > |R_i|$, no prediction target is assigned.

For each output token $\langle \mathrm{o} \rangle$, the prediction target is defined as the first token of the next reasoning step, i.e., r_1^{i+1}. No prediction target is assigned to memory tokens $\langle \mathrm{m} \rangle$; memory token parameters are learned solely through back propagations of other tokens.

Position ID Assignment. We assign dedicated position IDs (PIDs) to different token types in the sequence $\mathcal{S}$. Let ρ_t^i denote the PID of reasoning

[4] Given compression ratio c, each group contains c tokens except possibly the last group.

[5] We provide a detailed algorithmic description of this inference pipeline in Appendix A.

[6] When $d' = 0$, the foresight token reduces to standard next-token prediction.

token r_t^i. The output token $\langle \mathrm{o} \rangle$ after reasoning step $\tilde{R}_i$ is assigned PID $\rho^i_{|R_i|} + 1$, such that reasoning and output tokens occupy consecutive positions in the augmented CoT sequence. As shown in Figure 3, the $\langle \mathrm{o} \rangle$ is assigned PID 4 following r_3^1.

- **Memory Tokens.** For memory tokens $M_i = [\langle \mathrm{m}_1 \rangle, \ldots, \langle \mathrm{m}_l \rangle]$, we uniformly interpolate their PIDs over the positional span of the reasoning tokens in step $\tilde{R}_i$ to align with the UPL formulation in Section 2.1.2. For example, in Figure 3, reasoning step $\tilde{R}_1$ spans PIDs 1–3. With compression ratio $c = 3$, the memory token is assigned an interpolated PID 2.
- **Foresight Tokens.** For the foresight token $\langle \mathrm{f} \rangle$ inserted before r_t^i, its PID is assigned as $\rho^i_{t+d'-1}$, i.e., the PID immediately preceding the prediction target $r^i_{t+d'}$ in $\tilde{R}_i$. This positional alignment encourages $\langle \mathrm{f} \rangle$ to predict $r^i_{t+d'}$. For example, when $d' = 1$, the foresight token before r_1^1 shares the same PID as r_1^1, encouraging it to predict r_2^1.

2.3 Joint Training Framework

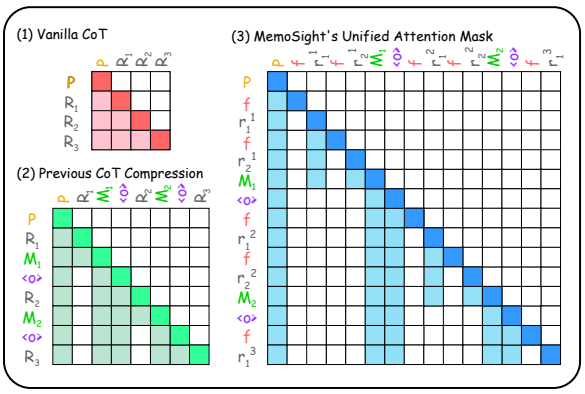

Figure 4: Comparison of training attention masks. (1) **Vanilla CoT** uses a standard causal mask. (2) In **LightThinker**, the attention restricts to memory tokens $\langle \mathrm{m} \rangle$ and current reasoning tokens, discarding historical reasoning steps. (3) **MemoSight** follows LightThinker to discard historical reasoning steps in its attention but further customizes the attention patterns due to the foresight token introduction.

Attention Mask Strategy. A critical component of our training framework is the specialized attention mask illustrated in Figure 4. Our attention design builds upon LightThinker while introducing additional masking rules for foresight tokens.

- **Compression Attention.** For the current reasoning step $\tilde{R}_i$, standard reasoning tokens r_t^i and memory tokens M_i follow standard causal attention over preceding reasoning tokens. For previous reasoning steps, they attend exclusively to the prompt P, condensed memory tokens $M_{<i}$, and output tokens, while the verbose reasoning traces $\tilde{R}_{<i}$ are masked out. This design compels the model to rely entirely on compressed memory for historical context.
- **MTP Attention.** A foresight token $\langle \mathrm{f} \rangle$ preceding r_t^i shares an identical attention pattern to r_t^i, attending to P, $M_{<i}$, $\langle \mathrm{o} \rangle$, the prefix reasoning tokens in the current step $\tilde{R}_i$, and itself. Its hidden state is masked from all future tokens, enabling foresight tokens to function as parallel predictive branches without interfering with the main reasoning process.

Joint Optimization Objective. A standard next-token prediction loss, $\mathcal{L}_{\mathrm{NTP}}$, is applied to reasoning tokens and output tokens $\langle \mathrm{o} \rangle$, while the loss on memory tokens $\langle \mathrm{m} \rangle$ is masked out. The multi-token prediction loss, $\mathcal{L}_{\mathrm{MTP}}$, is applied to foresight tokens $\langle \mathrm{f} \rangle$ to predict their future counterparts. The overall training objective is:

$$\mathcal{L}_{\mathrm{MemoSight}} = \lambda \mathcal{L}_{\mathrm{NTP}} + (1 - \lambda) \mathcal{L}_{\mathrm{MTP}},$$

where λ controls the trade-off between standard next-token prediction and foresight prediction.

3 Experiments

3.1 Experimental Settings

Baselines. We evaluate MemoSight on two mainstream LLMs, Qwen2.5-7B (Hui et al., 2024) and Llama-3.1-8B (Dubey et al., 2024), against six representative baselines categorized into three groups: (1) CoT Models: The standard instruction model using CoT prompting, alongside the DeepSeek-R1 Distilled model (Distill-R1) (Guo et al., 2025) (2) Trained Reasoning Models: A Vanilla baseline and LightThinker (Zhang et al., 2025a), both initialized from a Distill-R1 model and further instruction-tuned on the Bespoke-Stratos-17k dataset, a training paradigm that MemoSight also follows. (3) Training-Free Acceleration: Two post-hoc acceleration methods H2O (Zhang et al., 2023) and SepLLM (Chen et al., 2025) applied directly to the Vanilla model, which attempt to retain important KV Cache states through heuristic strategies. Details for all baselines are provided in Appendix B.1.

Datasets and Metrics. We evaluate on four datasets covering the primary reasoning paradigms

Method	GSM8K			MMLU			GPQA			BBH			AVG.		
	Acc ↑	Speed ↑	Peak ↓	Acc ↑	Speed ↑	Peak ↓	Acc ↑	Speed ↑	Peak ↓	Acc ↑	Speed ↑	Peak ↓	Acc ↑	Speed ↑	Peak ↓
Qwen2.5-7B Series															
CoT	88.32	24.63	519	70.01	23.78	676	28.28	25.50	998	71.31	27.47	582	64.48	25.35	694
Distill-R1	60.58	24.82	522	31.26	21.49	1289	22.22	19.21	4383	51.31	29.35	905	41.34	23.72	1775
Vanilla	90.83	20.69	1559	65.92	18.6	2241	37.37	15.10	7184	81.01	24.32	2212	68.78	19.68	3299
+ H2O	91.05	13.85	1024	62.32	10.36	1024	20.20	9.96	**1024**	**75.76**	14.14	1024	62.33	12.08	**1024**
+ SepLLM	**91.28**	11.03	1024	59.20	10.97	1024	12.63	11.30	**1024**	69.70	14.27	1024	58.20	11.89	**1024**
LightThinker	87.26	22.56	**684**	60.66	22.58	**827**	37.37	22.38	1969	66.46	28.13	**980**	62.94	23.91	1115
MemoSight	89.84	**30.94**	745	**63.49**	**27.54**	900	**40.91**	**27.64**	2102	73.13	**32.26**	1059	**66.84**	**29.60**	1202
Llama3.1-8B Series															
CoT	64.52	21.41	497	60.18	19.23	698	24.75	21.40	2230	53.13	26.28	596	50.65	22.08	1005
Distill-R1	57.62	22.78	524	20.93	17.89	1611	30.81	16.40	5640	29.49	27.65	1146	34.71	21.18	2230
Vanilla	89.31	18.33	1702	71.08	16.26	2800	37.37	14.17	6824	78.18	21.63	2515	68.99	17.60	3460
+ H2O	**89.31**	11.65	1024	**69.52**	9.03	1024	23.23	8.51	**1024**	**81.62**	13.04	1024	65.92	10.56	**1024**
+ SepLLM	88.32	19.74	1024	64.85	19.02	1024	15.66	19.27	**1024**	72.32	24.12	1024	60.29	20.54	**1024**
LightThinker	85.82	21.38	**664**	60.86	20.53	**932**	**37.37**	19.89	1933	72.93	26.11	999	64.25	21.98	1132
MemoSight	87.19	**27.91**	750	64.07	**25.89**	933	35.86	**26.67**	1942	78.38	**33.45**	**866**	**66.38**	**28.48**	1123

Table 1: Main results on the Qwen2.5-7B and Llama3.1-8B models. Acceleration methods are highlighted in blue, with **bold** and underlined values indicating the best and second-best results among them. Peak and Speed denote maximum context token counts and number of tokens generated per second, respectively.

of LLMs: mathematical reasoning GSM8K (Cobbe et al., 2021), algorithmic and compositional reasoning BBH (Suzgun et al., 2023), knowledge-based reasoning MMLU (Hendrycks et al., 2020), and expert-level scientific reasoning GPQA (Rein et al., 2024). We assess the models from two perspectives: task performance and computational efficiency. Task performance is measured by accuracy (Acc). Computational efficiency is quantified by peak context tokens (Peak) indicating memory footprint and number of tokens generated per second (Speed) reflecting inference throughput.

Implementation Details. All models are trained for 5 epochs with a batch size of 64. We fix the combined budget for reasoning and memory tokens at 4096 across all models. We set the NTP loss weight $\lambda = 0.7$, the compression ratio $c = 5$ (i.e., one memory token per five reasoning tokens) and the max foresight offset $d = 2$ (i.e., randomly sampling $d' \in \{0, 1, 2\}$ per training sample, which enables the parallel prediction of three tokens during inference). During evaluation, we employ greedy decoding with a maximum output length of 10240 for all models. We refer readers to Appendix B.2 for more details.

3.2 Main Results

Table 1 presents our main results. Consistent with prior findings (Li et al., 2025a), Distill-R1 underperforms CoT due to limited instruction-following capabilities. The Vanilla baseline, fine-tuned on reasoning data, exhibits strong task performance. Among the acceleration methods, MemoSight outperforms all the other methods across both the Qwen and Llama model families. For instance, on Qwen, MemoSight exceeds the second-best baseline, LightThinker, by an absolute 3.9 points in average accuracy. Furthermore, despite its acceleration design, MemoSight achieves average performance comparable to the Vanilla baseline, and even surpasses it on specific datasets (e.g., +3.54 on GPQA).

In terms of efficiency, training-free acceleration methods fail to improve inference speed over the Vanilla baseline.[7] In contrast, MemoSight surpasses both the Vanilla baseline and other acceleration methods, achieving the fastest overall inference. Specifically, MemoSight achieves an average speedup of 23.8% on Qwen and 29.6% on the Llama series compared to the context compression baseline, LightThinker. Regarding memory consumption, Table 1 shows that at a 5× compression ratio, MemoSight maintains a peak memory footprint comparable to LightThinker and well below the Vanilla baseline. Further analysis in Section 4.1 shows that MemoSight continues to outperform

[7]For example, we observe that H2O-generated answers are longer than those of the Vanilla baseline, as shown in Figure 5(c).

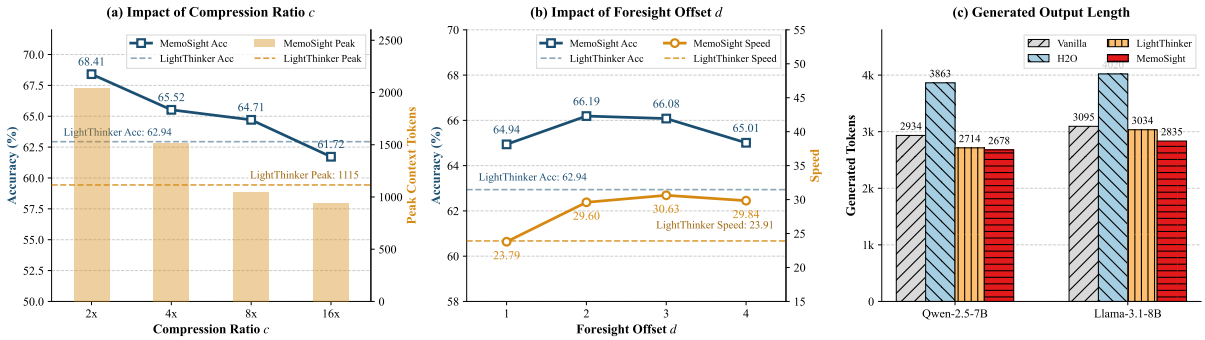

Figure 5: **Efficiency Analysis. (a) Compression Impact**: Accuracy and peak context token count under compression levels from $2\times$ to $16\times$. Higher compression reduces memory footprint but incurs accuracy degradation. **(b) Offset Impact**: Accuracy and inference speed with different foresight offsets ($d = 1$ to 4) setting; the same setting is used across training and inference.**(c) Average generated tokens** across all benchmarks for Vanilla, H2O, LightThinker, and MemoSight on Qwen-2.5-7B and Llama-3.1-8B. MemoSight generates the fewest tokens.

LightThinker even under lower memory budgets (i.e., higher compression ratio).

4 Analyses

4.1 Efficiency

In this section, we analyze MemoSight's efficiency and its tradeoff with reasoning performance.

How does the compression ratio c affect performance and efficiency? Context compression inherently trades off between memory efficiency and reasoning effectiveness. We evaluate MemoSight under compression ratios ranging from $2\times$ to $16\times$. As shown in Figure 5(a), higher compression ratios consistently reduce memory usage but also degrade accuracy. Compared to LightThinker, MemoSight provides a more favorable tradeoff between efficiency and performance. For example, under $8\times$ compression, MemoSight uses fewer peak tokens than LightThinker while achieving higher average accuracy (+1.77 points). However, aggressive compression at $16\times$ leads to a clear performance drop, suggesting that excessive compression loses critical reasoning information. Table 1 shows that, under a practical $5\times$ compression ratio, MemoSight reduces peak memory usage by 66% compared to Vanilla decoding with less than a 3% drop in average accuracy. In Appendix C.1, we further show that the reduced memory footprint improves inference speed on long sequences.

How does the MTP offset d affect performance and efficiency? The parameter d determines how far ahead the MTP module is trained to predict and thus controls the maximum number of parallel predictions enabled during inference. Figure 5(b) shows the impact of different MTP offsets on accuracy and inference speedup. MemoSight consistently outperforms LightThinker in both metrics across all settings by using MTP. MemoSight achieves most accuracy improvements at $d = 2$ (66.19%) and remains comparable at $d = 3$ (66.08%); smaller and larger offsets lead to less performance improvements, with $d = 1$ achieving 64.94% and $d = 4$ achieving 65.01%. These results suggest that a moderate look-ahead distance provides the most effective training signal to improve model's reasoning capability. Configurations with 2–4 foresight tokens achieve similar inference speedups. Based on these results, we use $d = 2$ as the default setting.

Does MemoSight generate more tokens than other methods? Zhang et al. (2025a) analyze the generated token lengths of different compression methods and find that LightThinker produces the shortest outputs among existing approaches. We perform the same analysis for MemoSight. Figure 5(c) reports the average number of generated tokens for Vanilla, H2O, LightThinker, and MemoSight across four datasets. Compared to LightThinker, MemoSight further reduces the generated token length and consistently produces the fewest tokens among all methods. Compared to Vanilla decoding, MemoSight reduces the average number of generated tokens by 9% on the Qwen model and 8% on the Llama model.

4.2 Ablation Study

We further ablate the core components of MemoSight, including Uniform Position Layout (UPL) which assigns uniformly interpolated position IDs

Method	GSM8K	MMLU	GPQA	BBH	Avg.
MemoSight	89.84	**63.49**	**40.91**	**73.13**	**66.84**
w/o UPL	88.10	59.49	39.90	67.47	63.74
w/o MTP	**89.99**	60.86	37.37	71.72	64.99
w/o AMA	86.96	59.10	32.32	69.90	62.07

Table 2: Ablation study of MemoSight's core components, including Uniform Position Layout (UPL), Adaptive Memory Allocation (AMA), and foresight-token-based MTP. Removing any component consistently degrades performance.

Method	Param.	GSM8K	MMLU	GPQA	BBH	Avg.
MemoSight	**4K**	**89.84**	**63.49**	**40.91**	**73.13**	**66.84**
MTP Middle	103M	88.17	61.25	38.89	69.49	64.45
MTP Last	103M	85.06	61.34	32.83	71.11	62.59

Table 3: Comparison between MemoSight and traditional MTP variants. Traditional MTP modules introduce more parameters but achieve lower accuracy.

to memory tokens over the compressed span, Adaptive Memory Allocation (AMA) which scales the number of memory tokens according to the reasoning step length; and foresight-token-based MTP. Detailed ablation settings are provided in Appendix B.3. We evaluate all variants on Qwen and report the results in Table 2.

The full model achieves the highest average accuracy (66.84), while removing any component consistently degrades performance. Among the components, AMA is particularly important for the knowledge-intensive GPQA benchmark, where removing AMA causes a large accuracy drop (-8.59). In contrast, removing UPL leads to the largest degradation on BBH (-5.66). Finally, removing MTP decreases the average accuracy by 1.85, indicating that the foresight-token mechanism improves reasoning performance in addition to accelerating inference.

4.3 Comparison with Traditional MTP

We vary the MTP mechanisms used in MemoSight to study their effect on accuracy. Specifically, we integrate DeepSeek-V3's MTP variant (MTP Last) (Liu et al., 2024) and a middle-layer variant (MTP Middle) (Wang et al., 2025) into MemoSight for comparison. Detailed descriptions of these variants are provided in Appendix B.4. Table 3 suggests that combining MTP with context compression is non-trivial. Both traditional MTP variants degrade performance compared to the MTP-free baseline (i.e., MemoSight w/o MTP in Table 2). We attribute this behavior to the large parameter overhead of traditional MTP modules (100M+ parameters), which makes joint optimization with context compression more difficult. In contrast, MemoSight unifies MTP and compression within a special-token-based framework, enabling joint optimization with only 4K additional parameters.

5 Related Work

To tackle CoT computational efficiency issues, recent studies CoConut (Hao et al., 2025) and its variants CODI (Shen et al., 2025) and SIM-CoT (Wei et al., 2025) propose latent reasoning that performs reasoning in the high dimensional latent space. However, latent reasoning lacks interpretability and is prone to learning dataset artifacts in its reasoning (Zhang et al., 2025b). As an alternative, LightThinker (Zhang et al., 2025a) explicitly compresses CoT into its discrete memory tokens step-by-step. MemoSight follows LighThinker in its design but further equips its memory tokens with a special position layout inspired by (Zhao et al., 2024) that enhances its compression capability a priori.

To the best of our knowledge, we are the first to combine MTP with CoT compression for CoT inference acceleration. We empirically show that such combination is non-trivial as most existing MTP frameworks require architectural modifications to the LLM (Liu et al., 2024; Gloeckle et al., 2024), which hinder performance when combined with CoT compression. Instead, in MemoSight, we propose to integrate MTP via a unified way through special tokens and a tailored position layout following (Gerontopoulos et al., 2025) — the same mechanism that we integrate CoT compression and demonstrate not only accelerated inference but also improved performance. We provide more comprehensive discussion of related work in Appendix D.

6 Conclusion

We present MemoSight to address memory and latency bottlenecks in Chain-of-Thought reasoning. While naively combining context compression and multi-token prediction degrades reasoning performance, MemoSight overcomes this through a unified design using memory tokens and foresight tokens, both of which equipped with a dedicated position layout. Compared to the CoT fine-tuned LLM counterpart, MemoSight reduces the KV cache footprint by 66% and improves inference speed by 56% with less than 3% answer accuracy degradation, reaching a new state-of-the-art efficiency-performance balance.

Limitations

Despite its effectiveness, MemoSight has several limitations both on CoT compression and MTP.

Compared to CoT prompting, MemoSight enables longer CoT reasoning by effectively shortening each reasoning step. However, we remark that MemoSight does not enable infinite long reasoning, as memory tokens still accumulate within the KV cache during each CoT step. We believe techniques beyond compression, such as memory management (Zhu et al., 2026) is required to achieve infinite long reasoning. MemoSight experiences performance degradation when pushed to high compression ratios where $16\times$ compression ratio already significantly degrades average accuracy in our experiments.

We integrate MTP into our unified framework and achieve both performance improvement and inference acceleration. However, we notice that our inference acceleration is only 24% for Qwen and 30% for Llama. Our analysis shows that even by training on more parallel predictions with larger d, the acceleration does not further improve as more futuristic tokens are mostly rejected through verification. It remains question how to better train MTP to learn more future token predictions in parallel reliably.

References

Daman Arora and Andrea Zanette. 2025. Training language models to reason efficiently. *arXiv preprint arXiv:2502.04463*.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. 2024a. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and 1 others. 2024b. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*.

Kaiyan Chang, Songcheng Xu, Chenglong Wang, Yingfeng Luo, Xiaoqian Liu, Tong Xiao, and Jingbo Zhu. 2024. Efficient prompting methods for large language models: A survey. *arXiv preprint arXiv:2404.01077*.

Guoxuan Chen, Han Shi, Jiawei Li, Yihang Gao, Xiaozhe Ren, Yimeng Chen, Xin Jiang, Zhenguo Li, Weiyang Liu, and Chao Huang. 2025. SepLLM: Accelerate large language models by compressing one segment into one separator. In *Forty-second International Conference on Machine Learning*.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. 2023. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3829–3846.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. 2023. In-context autoencoder for context compression in a large language model. *arXiv preprint arXiv:2307.06945*.

Anastasios Gerontopoulos, Spyros Gidaris, and Nikos Komodakis. 2025. Multi-token prediction needs registers. *arXiv preprint arXiv:2505.10518*.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. 2024. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*.

Daya Guo, Dejian Yang, Haowei Zhang, Song, and 1 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2025. Training large language models to reason in a continuous latent space. *Preprint*, arXiv:2412.06769.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Haoxiang Jia, Earl T Barr, and Sergey Mechtaev. 2026. Compressing code context for llm-based issue resolution. *arXiv preprint arXiv:2603.28119*.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 13358–13376.

Xiaomin Li, Zhou Yu, Zhiwei Zhang, Xupeng Chen, Ziji Zhang, Yingying Zhuang, Narayanan Sadagopan, and Anurag Beniwal. 2025a. When thinking fails: The pitfalls of reasoning for instruction-following in llms. *arXiv preprint arXiv:2505.11423*.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024a. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024b. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*.

Zongqian Li, Yixuan Su, and Nigel Collier. 2025b. 500xcompressor: Generalized prompt compression for large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 25081–25091.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Xin Liu, Runsong Zhao, Pengcheng Huang, Xinyu Liu, Junyi Xiao, Chunyang Xiao, Tong Xiao, Shengxiang Gao, Zhengtao Yu, and Jingbo Zhu. 2025. Autoencoding-free context compression for llms via contextual semantic anchors. *arXiv preprint arXiv:2510.08907*.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, and 1 others. 2024. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. *arXiv preprint arXiv:2403.12968*.

Jianhui Pang, Fanghua Ye, Derek Wong, Xin He, Wanshun Chen, and Longyue Wang. 2024. Anchor-based large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 4958–4976.

Weizhen Qi, Yu Yan, Yeyun Gong, Dayiheng Liu, Nan Duan, Jiusheng Chen, Ruofei Zhang, and Ming Zhou. 2020. Prophetnet: Predicting future n-gram for sequence-to-sequencepre-training. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2401–2410.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *First conference on language modeling*.

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*.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and 1 others. 2023. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051.

Jiwei Tang, Shilei Liu, Zhicheng Zhang, Qingsong Lv, Runsong Zhao, Tingwei Lu, Langming Liu, Haibin Chen, Yujin Yuan, Hai-Tao Zheng, and 1 others. 2026a. Read as human: Compressing context via parallelizable close reading and skimming. *arXiv preprint arXiv:2602.01840*.

Jiwei Tang, Shilei Liu, Zhicheng Zhang, Yujin Yuan, Libin Zheng, Wenbo Su, and Bo Zheng. 2026b. Comi: Coarse-to-fine context compression via marginal information gain. *arXiv preprint arXiv:2602.01719*.

Jiwei Tang, Jin Xu, Tingwei Lu, Zhicheng Zhang, YimingZhao YimingZhao, LinHai LinHai, and Hai-Tao Zheng. 2025a. Perception compressor: A training-free prompt compression framework in long context scenarios. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 4093–4108.

Jiwei Tang, Zhicheng Zhang, Shunlong Wu, Jingheng Ye, Lichen Bai, Zitai Wang, Tingwei Lu, Lin Hai, Yiming Zhao, Hai-Tao Zheng, and 1 others. 2025b. Gmsa: Enhancing context compression via group merging and layer semantic alignment. *arXiv preprint arXiv:2505.12215*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Jianjin Wang, Runsong Zhao, Xiaoqian Liu, Yuan Ge, Ziqiang Xu, Tong Xiao, Shengxiang Gao, Zhengtao Yu, and Jingbo Zhu. 2025. Mtp-s2ut: Enhancing speech-to-speech translation quality with multi-token prediction. *arXiv preprint arXiv:2510.10003*.

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.

Jintian Zhang, Yuqi Zhu, Mengshu Sun, Yujie Luo, Shuofei Qiao, Lun Du, Da Zheng, Huajun Chen, and Ningyu Zhang. 2025a. Lightthinker: Thinking step-by-step compression. *arXiv preprint arXiv:2502.15589*.

Peitian Zhang, Zheng Liu, Shitao Xiao, Ninglu Shao, Qiwei Ye, and Zhicheng Dou. 2024. Long context compression with activation beacon. *arXiv preprint arXiv:2401.03462*.

Yuyi Zhang, Boyu Tang, Tianjie Ju, Sufeng Duan, and Gongshen Liu. 2025b. Do latent tokens think? a causal and adversarial analysis of chain-of-continuous-thought. *arXiv preprint arXiv:2512.21711*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Re, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Runsong Zhao, Shilei Liu, Jiwei Tang, Langming Liu, Haibin Chen, Weidong Zhang, Yujin Yuan, Tong Xiao, Jingbo Zhu, Wenbo Su, and 1 others. 2026a. Comet: Collaborative memory transformer for efficient long context modeling. *arXiv preprint arXiv:2602.01766*.

Runsong Zhao, Shilei Liu, Jiwei Tang, Langming Liu, Haibin Chen, Weidong Zhang, Yujin Yuan, Tong Xiao, Jingbo Zhu, Wenbo Su, and 1 others. 2026b. Poc: Performance-oriented context compression for large language models via performance prediction. *arXiv preprint arXiv:2603.19733*.

Runsong Zhao, Xin Liu, Xinyu Liu, Pengcheng Huang, Chunyang Xiao, Tong Xiao, and Jingbo Zhu. 2024. Position ids matter: An enhanced position layout for efficient context compression in large language models. *arXiv preprint arXiv:2409.14364*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, and 1 others. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2).

Yiqing Zhou, Yu Lei, Shuzheng Si, Qingyan Sun, Wei Wang, Yifei Wu, Hao Wen, Gang Chen, Fanchao Qi, and Maosong Sun. 2025. From context to edus: Faithful and structured context compression via elementary discourse unit decomposition. *arXiv preprint arXiv:2512.14244*.

Yuqi Zhu, Jintian Zhang, Zhenjie Wan, Yujie Luo, Shuofei Qiao, Zhengke Gui, Da Zheng, Lei Liang, Huajun Chen, and Ningyu Zhang. 2026. Lightthinker++: From reasoning compression to memory management. *arXiv preprint arXiv:2604.03679*.

A Inference Procedure

We outline the iterative inference procedure in Algorithm 1.

Algorithm 1 MemoSight Iterative Inference

Require: Prompt P, model $\mathcal{M}$ with KV cache S, foresight offset d, compression rate c, special tokens $\langle \text{e} \rangle$ (segment boundary) and $\langle \text{eos} \rangle$, draft tokens $\hat{R}$, and accepted tokens A.

$S \leftarrow \text{Prefill}(P)$ // Initialize KV cache
while $\langle \text{eos} \rangle \notin A$ **do**
  $R_{\text{current}} \leftarrow \emptyset$ // Initialize segment tokens
  // Phase 1: Foresight-based Acceleration
  repeat
    $A \leftarrow \emptyset$ // Initialize accept tokens
    $\hat{R}, A \leftarrow \text{SpeculativeStep}(\mathcal{M}, S, d)$
    $S \leftarrow S \oplus A$ // Append to cache
    $R_{\text{current}} \leftarrow R_{\text{current}} \oplus A$ // Accumulate segment
  until $\langle \text{e} \rangle \in A$ **or** $\langle \text{eos} \rangle \in A$
  // Phase 2: Dynamic Compression
  if $\langle \text{e} \rangle \in A$ **then**
    $l \leftarrow \lceil |R_{\text{current}}|/c \rceil$
    $M \leftarrow \text{Forward}(\mathcal{M}, \langle \text{m}_1, \ldots, \text{m}_l, \text{o} \rangle)$
    $S \leftarrow \text{UpdateCache}(S, R_{\text{current}}, M)$ // Evict verbose and insert memory
  end if
end while

B Experimental Details

B.1 Baseline Details

During evaluation, we employ greedy decoding with a maximum output length of 10,240 tokens for all models. We compare MemoSight against the following baselines:

- **CoT**: A baseline that applies few-shot Chain-of-Thought (CoT) prompting to the Qwen2.5-7B (Hui et al., 2024) and Llama-3.1-8B (Dubey et al., 2024) models without additional training.
- **Distill-R1** (Guo et al., 2025): A reasoning model distilled from DeepSeek-R1's response data, built upon the Qwen and Llama architectures.
- **Vanilla**: A standard full-parameter instruction-tuned model. Operating without any compression or acceleration mechanisms, it serves as the empirical upper bound for reasoning accuracy.
- **H2O** (Zhang et al., 2023): A training-free KV cache eviction strategy that retains "Heavy Hitter" tokens and a local window. We apply H2O to the Vanilla model using a sliding window of 1024 and a heavy-hitter budget of 512 tokens.
- **SepLLM** (Chen et al., 2025): A training-free framework that preserves KV caches for initial tokens, separators, and a local window. We configure SepLLM with an initial cache size of 384, a separator budget of 64, and a local window of 256, maintaining a total cache capacity of 1024.
- **LightThinker** (Zhang et al., 2025a): A post-training method that compresses each reasoning step into a fixed number of memory tokens. Upon reaching a step boundary, the model generates memory tokens to summarize the context, after which the original reasoning tokens are evicted from the KV cache.

B.2 Training Details

The Vanilla baseline, LightThinker and MemoSight are initialized from DeepSeek-R1-Distill (Guo et al., 2025) and trained on the Bespoke-Stratos-17k (BS17K) dataset for 5 epochs. Experiments are conducted on 8 H200 GPUs using DeepSpeed ZeRO-3 offload. We use a micro-batch size of 2 and 4 gradient accumulation steps, yielding a global batch size of 64. We employ a cosine learning rate schedule with a 0.05 warmup ratio. The peak learning rate for Vanilla is set to 1e-5, while for the other models, it is set to 2e-5.

All the CoT token length (including memory tokens) has been set to 4096; to accommodate the additional foresight tokens introduced during training, MemoSight is extended to a maximum sequence length of 8192. For MemoSight specifically, we set the compression ratio to $c = 5$ and randomly sample the foresight offset $d' \in \{0, 1, 2\}$ during data construction. The standard LM loss and the MTP loss are weighted at 0.7 and 0.3, respectively (i.e., $\lambda = 0.7$).

B.3 Ablation Details

This section describes the experimental setup for our ablation studies, which evaluate the contributions of Uniform Position Layout (UPL), Adaptive Memory Allocation (AMA), and foresight-token-based multi-token prediction (MTP).

- **MemoSight w/o UPL:** This variant assigns monotonically increasing position IDs to memory tokens, rather than uniformly interpolated PIDs. Foresight tokens are assigned PIDs following the standard offset rule (i.e., $t + d - 1$ when predicting the token at $t + d$).
- **MemoSight w/o AMA:** Instead of using adaptive memory allocation with a fixed compression ratio, this variant allocates a fixed budget of 9

memory tokens per reasoning step, following the LightThinker configuration.

- **MemoSight w/o MTP:** This variant removes foresight tokens and the multi-token prediction objective, relying solely on memory tokens for context compression.

All ablation experiments are conducted on the Qwen series models. Training settings are kept identical across all variants and follow the same configuration as MemoSight and LightThinker.

B.4 Traditional MTP Details

To assess the effectiveness of our foresight-token-based MTP, we establish traditional MTP baselines operating under identical context compression settings. Following DeepSeek-V3 (Liu et al., 2024), the standard configuration sequentially appends two transformer blocks to the final layer's hidden states, predicting the next two tokens via a shared LM head. Furthermore, inspired by Wang et al. (2025), who demonstrated that leveraging intermediate representations yields superior performance in speech tasks, we evaluate an additional variant that feeds intermediate layer hidden states—rather than the final layer's output—into the MTP module.

B.5 Evaluation Details

During inference, we evaluate all models using greedy decoding with a repetition penalty of 1.1. Prompt configurations for Table 1 are as follows: Vanilla, H2O, SepLLM, LightThinker, and MemoSight share the same system prompt (Figure 11) and task prompts (Figure 12). Distill-R1 uses these task prompts but omits the system prompt. The CoT baseline uses a few-shot system prompt (Figure 9) alongside task-specific prompts for different benchmarks (Figure 10). For MMLU (Hendrycks et al., 2020) and GPQA (Rein et al., 2024), multiple-choice options are randomized to prevent positional bias.

C More Analyses and Discussions

C.1 Long-Context Efficiency Analysis

Figure 6 compares the peak memory and inference time of MemoSight against the vanilla baseline during autoregressive generation. We set a fixed compression ratio of 7, compressing every 56 tokens. Results show that context compression effectively reduces memory consumption, lowering peak memory by over 80%. Regarding inference speed, since KV cache reduction primarily accelerates the attention layer rather than other modules, the speed improvement only becomes evident when the context length exceeds 8k.

C.2 Case Study

Figure 8 illustrates how preserving intermediate reasoning states prevents cascading errors in multi-step calculations. In this nutritional calculation problem, the baseline model, LightThinker, correctly computes the remaining calorie budget but skips calculating the grams per serving. As a result, it computes an incorrect energy density by dividing the per-serving calories by the whole-bag weight ($250 \div 300$), leading to a flawed self-verification and an incorrect final answer of 120 g. In contrast, MemoSight explicitly preserves all intermediate states. It calculates the correct serving size ($300 \div 5 = 60$) before applying the calorie constraint. This ensures the final scaling step is grounded in the correct premise ($(200 \div 250) \times 60$), allowing the model to consistently arrive at the correct answer of 48 g.

C.3 Impact of Loss Weights

We investigate the effect of the loss weight distribution by comparing two configurations: a higher weight on the standard Language Modeling (LM) loss ($\lambda = 0.7$) and an equal weighting ($\lambda = 0.5$). As illustrated in Figure 7, the $0.7/0.3$ configuration consistently outperforms the equal weighting baseline across various compression ratios (c). Notably, allocating a larger weight to the MTP objective (0.5) degrades average accuracy, suggesting that excessive planning supervision interferes with the primary generation task. Conversely, a moderate MTP weight (0.3) achieves a more effective balance, preserving core reasoning capabilities while still benefiting from the lookahead training signal.

D Related Work

D.1 Context Compression

Context compression mitigates the computational and memory overhead of long-sequence processing by condensing contexts while preserving semantic information. Since our work focuses on long-form generation rather than prompt prefilling, we exclude methods that primarily compress the input context during the prefilling stage, such as AutoCompressor (Chevalier et al., 2023),

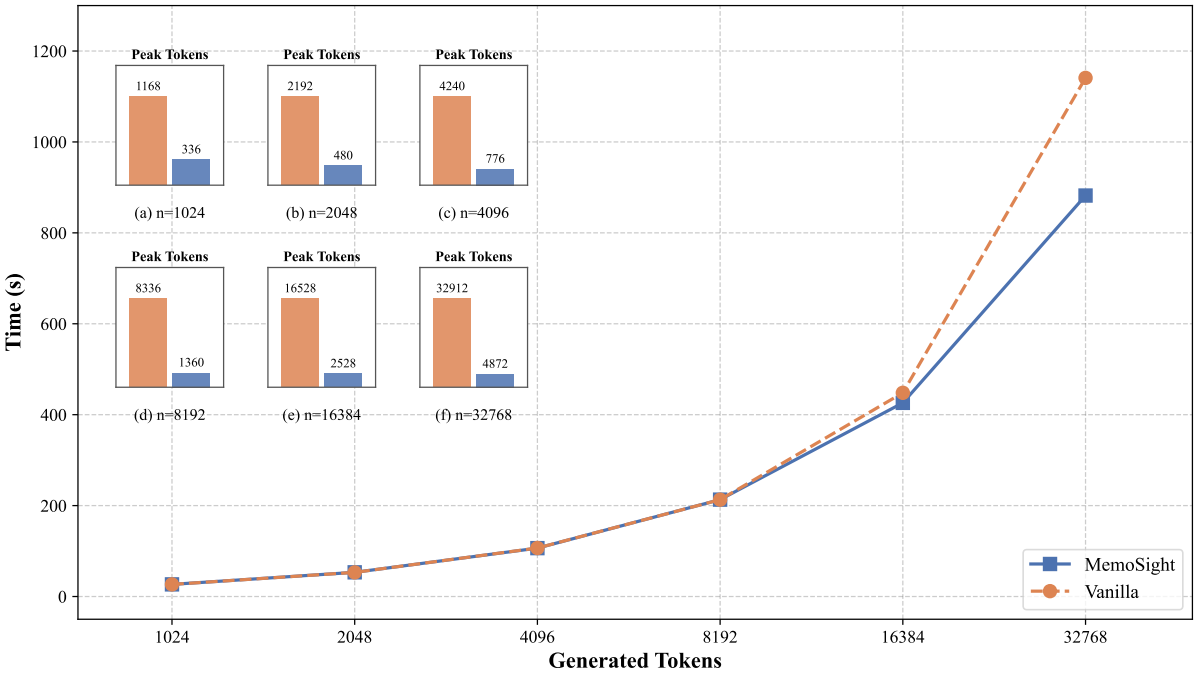

Figure 6: Time and memory efficiency evaluation. The main plot shows the inference time of MemoSight and Vanilla as the number of generated tokens increases. The inset charts (a)-(f) compare the peak token usage at different sequence lengths.

ICAE (Ge et al., 2023), LLMLingua (Jiang et al., 2023), 500xCompressor (Li et al., 2025b), ActivationBeacon (Zhang et al., 2024), PyramidKV (Cai et al., 2024b), Perception Compressor (Tang et al., 2025a), SnapKV (Li et al., 2024a), GMSA (Tang et al., 2025b), PoC (Zhao et al., 2026b), COMI (Tang et al., 2026b), RAM (Tang et al., 2026a), SAC (Liu et al., 2025), and CoMeT (Zhao et al., 2026a).

Generation-phase context compression methods generally fall into three paradigms: latent reasoning, explicit token selection, and implicit latent condensation.

1) Latent reasoning. To avoid the verbosity of explicit chain-of-thought (CoT) reasoning, methods such as Coconut (Hao et al., 2025) and Soft-CoT (Xu et al., 2025) perform reasoning in continuous latent spaces. To improve training stability and mitigate latent collapse, CODI (Shen et al., 2025) aligns latent states with natural-language CoT trajectories, while SIM-CoT (Wei et al., 2025) introduces auxiliary step-level supervision. However, Zhang et al. (2025b) argue that current latent reasoning models often exhibit pseudo-reasoning behaviors rather than genuine reasoning capabilities.

2) Explicit token selection. Early approaches remove tokens according to importance metrics, such as LLMLingua (Jiang et al., 2023; Pan et al., 2024). However, aggressive token pruning may damage local coherence and logical dependencies. More recent methods incorporate structural information during compression. For example, Context Compressor (Zhou et al., 2025) leverages discourse trees to preserve global structure, SWEzze (Jia et al., 2026) extracts minimal sufficient subsequences for code repositories, and TokenSkip (Xia et al., 2025) dynamically omits redundant reasoning tokens during decoding.

3) Implicit latent condensation. This paradigm compresses contexts into continuous latent embeddings, commonly represented as memory tokens. AnLLM (Pang et al., 2024) employs anchor-based self-attention to distill sequence information into specialized anchor tokens. LightThinker (Zhang et al., 2025a) compresses intermediate reasoning traces into compact gist tokens and discards verbose reasoning states to reduce KV cache growth during generation. Building upon this idea, LightThinker++ (Zhu et al., 2026) introduces adaptive memory management through memory primitives and trajectory synthesis, enabling stable KV cache usage in long-horizon reasoning tasks. Unlike prefilling-based compression methods, these approaches explicitly target the incremental memory overhead arising during autoregressive generation.

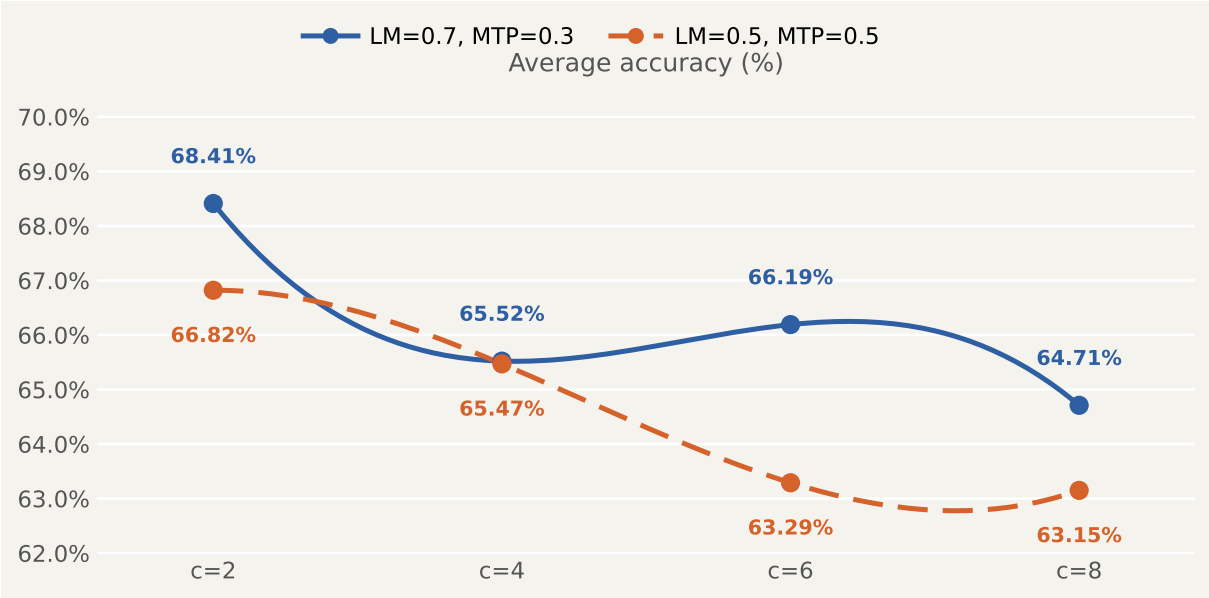

Figure 7: Impact of loss weight configuration (λ) on average accuracy across varying compression ratios (c). The blue solid line represents a higher weight on the standard LM loss ($\lambda = 0.7$), while the orange dashed line represents an equal weighting ($\lambda = 0.5$).

D.2 Multi-Token Prediction

Multi-token prediction (MTP) extends the standard next-token prediction objective by simultaneously predicting multiple future tokens, thereby densifying training signals and encouraging longer-horizon planning. Early work such as ProphetNet (Qi et al., 2020) introduced MTP for sequence-to-sequence learning, but its multi-stream attention mechanism scales poorly to large language models.

Recent studies address this limitation using auxiliary prediction heads. Gloeckle et al. (2024) employ parallel decoding heads to improve generative modeling, while Liu et al. (2024) introduce sequential prediction heads to enhance implicit planning within hidden states. However, these gains are mainly observed during large-scale pretraining. To better support fine-tuning scenarios, Gerontopoulos et al. (2025) recently proposed a special-token-based MTP formulation.

To the best of our knowledge, our work is the first to integrate multi-token prediction with context compression, further accelerating reasoning through speculative decoding (Cai et al., 2024a; Li et al., 2024b).

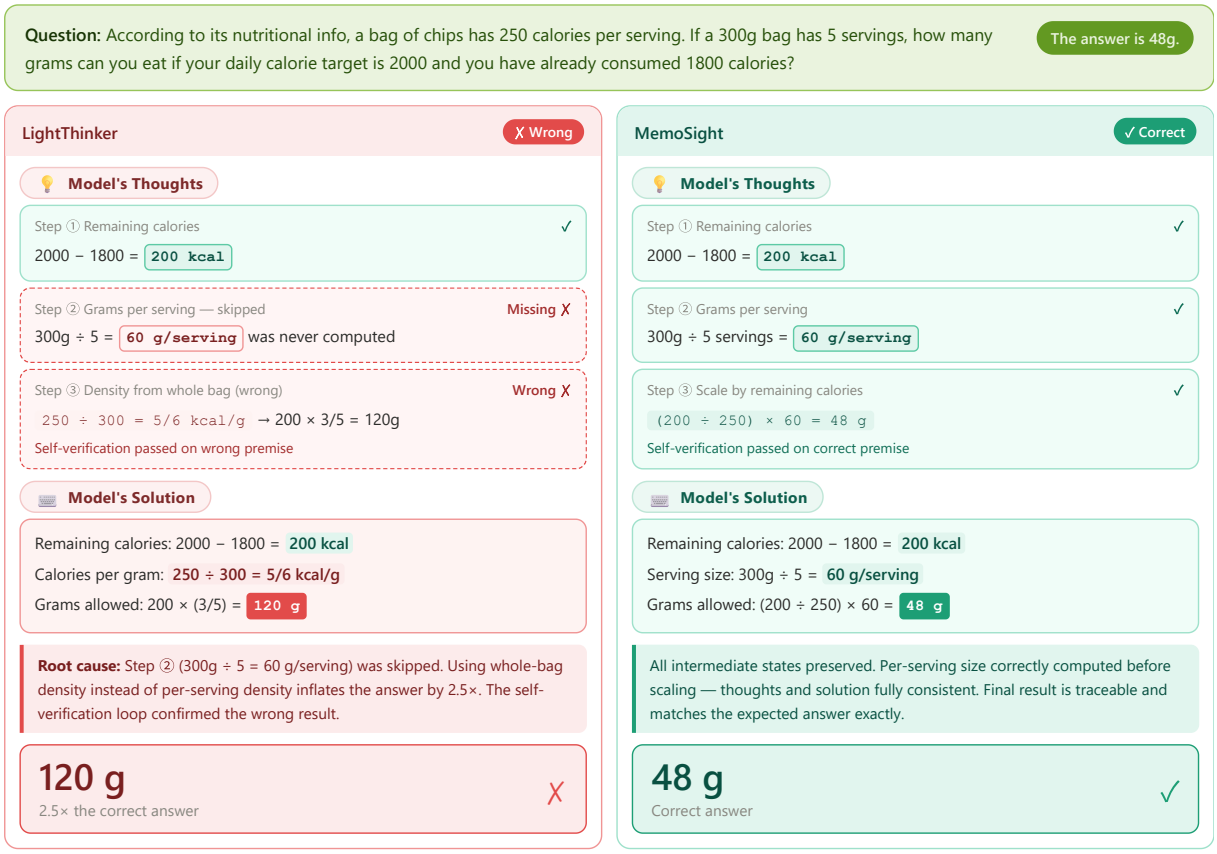

Figure 8: Case Study comparing the reasoning trajectories of LightThinker and MemoSight.

System Prompt:
Below is a question. Please think through it step by step, and then provide the final answer. If options are provided, please select the correct one.

Output format:
Use "<THOUGHT>...</THOUGHT>" to outline your reasoning process, and enclose the final answer in '\boxed{}'.

Example 1:
Question:
What is 2 + 3?
Output:
<THOUGHT>First, I recognize that this is a simple addition problem. Adding 2 and 3 together gives 5.</THOUGHT>
Therefore, the final answer is \boxed{5}.

Example 2:
Question:
What is 2 + 3?
A. 4
B. 5
C. 10

Output:
<THOUGHT>First, I recognize that this is a simple addition problem. Adding 2 and 3 together gives 5.</THOUGHT>
Therefore, the final answer is \boxed{B}.

Figure 9: System prompt for Qwen2.5-7B-Instruct and Llama3.1-8B-Instruct.

GSM8K:
Return your final response within \boxed{}. {Question}

MMLU:
Please select the option that best answers the question. Return your final response within \boxed{}. Here are the Question:
{Question}

GPQA:
Given a question, please select the option that best answers it. Return your final response within \boxed{}. {Question}

BBH:
Return your final response within \boxed{}. If options are provided, please select the correct one.
{Question}

Figure 10: Task prompt for Qwen2.5-7B-Instruct and Llama3.1-8B-Instruct.

System Prompt:
Your role as an assistant involves thoroughly exploring questions through a systematic long thinking process before providing the final precise and accurate solutions. This requires engaging in a comprehensive cycle of analysis, summarizing, exploration, reassessment, reflection, backtracking, and iteration to develop well-considered thinking process. Please structure your response into two main sections: Thought and Solution. In the Thought section, detail your reasoning process using the specified format: <|begin_of_thought|> {thought with steps separated with '\n\n'} <|end_of_thought|> Each step should include detailed considerations such as analyzing questions, summarizing relevant findings, brainstorming new ideas, verifying the accuracy of the current steps, refining any errors, and revisiting previous steps. In the Solution section, based on various attempts, explorations, and reflections from the Thought section, systematically present the final solution that you deem correct. The solution should remain a logical, accurate, concise expression style and detail necessary steps needed to reach the conclusion, formatted as follows: <|begin_of_solution|> {final formatted, precise, and clear solution} <|end_of_solution|> Now, try to solve the following question through the above guidelines:

Figure 11: The shared system prompt applied to Vanilla, H2O, SepLLM, LightThinker, and MemoSight across the Qwen and Llama series.

GSM8K/MMLU/GPQA/BBH:
Return your final response within \boxed{}. {Question}

Figure 12: The shared task prompt applied to Vanilla, H2O, SepLLM, LightThinker, and MemoSight across the Qwen and Llama series.