REVISITING WHAT, WHERE, AND WHEN IN LATENT MEMORY GENERATION FOR FROZEN LLM REASONERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Latent memory generation offers a promising approach to augmenting the test-time reasoning of frozen large language models, in which an auxiliary Weaver synthesizes continuous memory representations and a Trigger selectively injects them into the reasoning stream. However, existing implementations rely on three convenient but weakly justified defaults: the Weaver is trained with brief rationales, latent memory is read from the final Transformer layer, and memory recall is considered only at punctuation boundaries. We revisit these choices through a unified *What–Where–When* framework. For **what to remember**, we propose CoT-Supervised Memory Compression (CSMC). CSMC first trains the Weaver with richer CoT supervision and then introduces task-driven compression during Trigger reinforcement learning: CoT supervision improves 15 of 16 evaluated configurations, while compression further improves eight using fewer memory vectors. For **where to extract**, we propose Depth-Aware Readout (DAR) and show that intermediate-layer readout outperforms final-layer readout in 11 configurations and matches it in one, with a macro-average gain of 4.50 percentage points. For **when to recall**, we introduce the Reasoning-State-Aware Trigger (RSAT), which replaces punctuation-based heuristics with predictive-entropy-based activation and improves StrategyQA accuracy from 63.17% to 76.13% on Qwen2.5-7B-Instruct. Together, these results recast latent memory generation not as a fixed architecture but as an optimizable design space spanning what information is distilled and retained, where memory is read out, and when it is recalled, providing a more systematic foundation for augmenting the reasoning of frozen language models. The code will be released at https://github.com/anonymous-researcher-coder/latent-memory.

1 INTRODUCTION

Enhancing the reasoning capabilities of Large Language Models (LLMs) without retraining the base model has emerged as a pivotal research challenge. While LLMs excel at static knowledge retrieval, they often struggle with tasks requiring multi-step reasoning, long-chain computation, or cross-paragraph synthesis. These failures typically stem not from a lack of knowledge, but from the difficulty of activating relevant information at the right moment during the inference process. To address this, recent works have introduced an auxiliary "memory pathway" for frozen LLMs (Wu et al., 2022; Packer et al., 2023; Zhang et al., 2025), enabling them to query information that is crucial for the current reasoning step but otherwise difficult to trigger in a standard forward pass. These mechanisms range from external retrieval systems (Jiang et al., 2023; Su et al., 2024) and embedding-level token caches (Wu et al., 2022; Liu et al., 2025) to the **Latent Memory Generation (LMG)** paradigm we focus on. LMG augments a frozen reasoner with a lightweight auxiliary module (**Weaver**) that synthesizes compact latent tokens and a controller (**Trigger**) that decides when to inject them, offering a practical middle ground between prompting-only methods and full model adaptation (Hao et al., 2024; Zhang et al., 2025).

Despite its promise, current LMG implementations rely on three convenient but weakly justified defaults, each corresponding to a fundamental design question. We revisit these choices through a unified *What–Where–When* framework, illustrated in Figure 1. Regarding **what** to remember,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

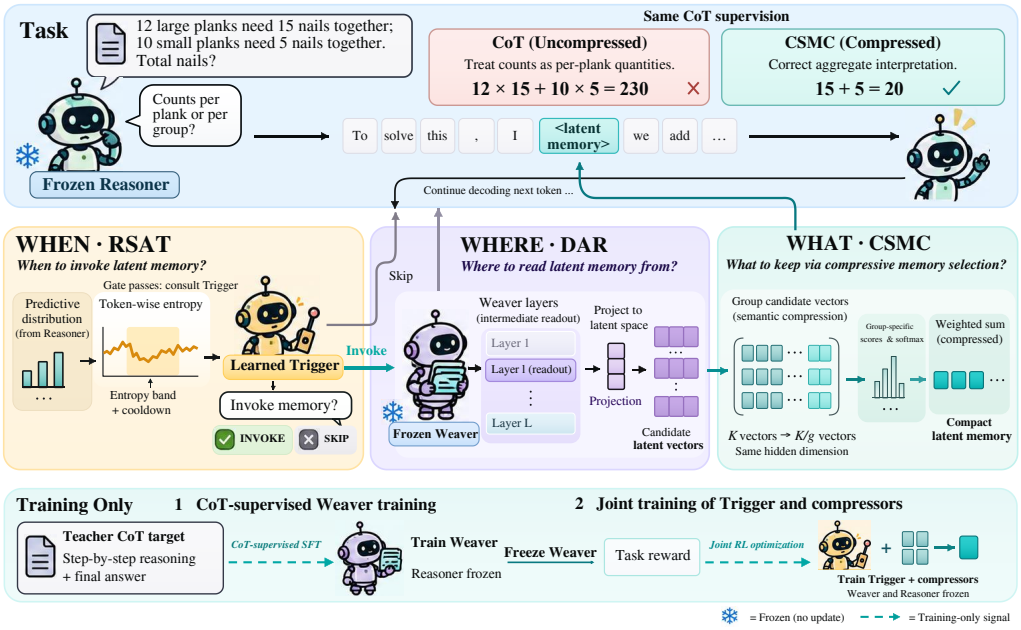


Figure 1: Overview of the What–Where–When framework.

native supervision may omit intermediate reasoning useful for memory generation, motivating richer CoT targets (Wei et al., 2022). To address this, we instead propose **CoT-Supervised Memory Compression (CSMC)**, which combines richer reasoning supervision with task-driven compression to preserve task-relevant information under a reduced memory budget. Regarding **where** to extract memory, the final layer is optimized for next-token prediction, whereas latent memories are re-injected to support subsequent reasoning. This difference motivates examining intermediate layers as alternative sources of memory representations. We therefore introduce **Depth-Aware Readout (DAR)**, which treats extraction depth as an explicit design variable and explores intermediate-layer readout for latent-memory generation. Finally, regarding **when** to recall, conventional punctuation-based triggering restricts memory-invocation opportunities to surface-level boundaries, potentially missing uncertain reasoning steps within a clause. To address this, we propose the **Reasoning-State-Aware Trigger (RSAT)**, which dynamically times memory injection based on the model's internal state. Specifically, we operationalize RSAT using the predictive entropy of the next-token distribution, leveraging the model's instantaneous uncertainty as a natural signal for when reasoning support is needed.

Across four backbones and four benchmarks, our results demonstrate the value of revisiting all three design choices. For *what*, CoT supervision improves accuracy in 15 of 16 configurations, while compression further improves eight using fewer memory vectors. For *where*, DAR improves SVAMP accuracy from 73.33% to 89.00% on Qwen3.5-4B, showing that intermediate-layer readout can provide more effective latent memory than final-layer readout. For *when*, RSAT with $[P_{80}, P_{95})$ achieves the best aggregate ranking among the evaluated bands and improves macro-average accuracy by 2.05 percentage points over punctuation-gated triggering.

These results further suggest that memory content and capacity, readout depth, and recall timing should be treated as optimizable design variables in latent memory generation rather than fixed engineering defaults. Our main contributions are as follows:

- **A Structured LMG Framework:** We introduce the What–Where–When perspective, treating memory content and capacity, readout depth and recall timing as design variables rather than fixed defaults.
- **CoT-Supervised Memory Compression (CSMC):** Addressing *what* to remember, we combine richer CoT supervision with task-driven compression to preserve useful reasoning information.

- **Depth-Aware Readout (DAR):** Addressing *where* to extract, we show that intermediate-layer representations can outperform conventional final-layer readout, making extraction depth an explicit design choice.
- **Reasoning-State-Aware Trigger (RSAT):** Addressing *when* to recall, we replace punctuation-based triggering with predictive-entropy-based activation to align memory recall with the reasoner's uncertainty.

2 PRELIMINARIES

A detailed discussion of related work is provided in Appendix B. We build on the latent memory generation framework of MemGen (Zhang et al., 2025), which augments a frozen reasoner with two lightweight LoRA-based modules: a *Weaver* that synthesizes latent memory tokens, and a *Trigger* that decides when those tokens should be invoked. Concretely, given an input task and the reasoner's autoregressive generation, the Trigger monitors the reasoner's evolving prefix and, at selected positions, outputs a binary decision $d_j \in \{\textsc{Invoke}, \textsc{Skip}\}$ over whether the Weaver should fire. Upon invocation, the Weaver, conditioned on the same prefix, produces a fixed-length latent memory sequence $\mathbf{M}_t \in \mathbb{R}^{K \times d_{\text{model}}}$, which is inserted into the reasoner's hidden-state stream so that subsequent token generation is conditioned on this enriched context. The frozen reasoner thus alternates between standard autoregressive decoding and short bursts of latent-memory-conditioned decoding, without any modification to its base parameters.

This framework leaves three implementation choices unspecified, each of which is filled in by a default whose adequacy has not been systematically examined. **What** the Weaver should encode is determined by supervised distillation from the dataset's native target field, whose rationale and final answer may not contain the reasoning information most useful for latent memory. **Where** the Weaver's output hidden state is taken from is fixed to the Weaver's final transformer layer, which is the representation already prepared for next-token prediction. **When** the Trigger should fire is constrained to punctuation boundaries: at any token-generation step, the Trigger is consulted only when the just-generated token is a delimiter (comma, period, or newline); at all other positions, no invocation is considered. These three defaults (raw-rationale supervision, last-layer readout, and punctuation-gated triggering) together define the configuration as instantiated in MemGen.

In what follows, we revisit each of these defaults in turn: Section 3.1 reconsiders *what* the Weaver is trained to encode, Section 3.2 reconsiders *where* its hidden state is extracted, and Section 3.3 reconsiders *when* the Trigger fires.

3 METHODS

We operationalize the What–Where–When framework through three targeted modifications, addressed in Sections 3.1–3.3. The remainder of the framework (the dual-module Weaver–Trigger architecture, the LoRA-based parameterization, the memory injection pathway, and the frozen base reasoner) is held strictly constant, so any change in performance can be attributed to the specific design variable under study.

3.1 COT-SUPERVISED MEMORY COMPRESSION (CSMC)

Motivation. The *what-to-remember* question concerns both the information used to train the Weaver and the amount of generated memory retained by the Reasoner. The original system supervises the Weaver with short, answer-oriented rationales and directly injects all generated memory vectors. Short supervision may provide insufficient reasoning information, whereas retaining every vector may introduce redundancy. We therefore propose CoT-Supervised Memory Compression (CSMC), which separates memory acquisition from memory consolidation.

CoT-Supervised Memory Acquisition. CSMC trains the Weaver with richer chain-of-thought (CoT) supervision. Each target consists of a detailed step-by-step reasoning trajectory followed by the final answer. This provides the Weaver with more complete reasoning information while preserving the original architecture and memory budget. The teacher-generated CoT is used only

during training. Once trained, the Weaver is frozen and produces candidate latent memories for subsequent compression.

Task-Driven Memory Compression. Given a candidate latent-memory sequence $\mathbf{M} = [\mathbf{m}_1, \ldots, \mathbf{m}_K]$, CSMC partitions it into contiguous, non-overlapping groups of size g. We use separately parameterized compressors for prompt and inference memory, each with a group-specific scoring head. For each group, the corresponding head produces importance scores, which are normalized with a softmax to compute a weighted combination:

$$\boldsymbol{\alpha}_q = \mathrm{softmax}\left(f_\phi^{(s,q)}\left([\mathbf{m}_{(q-1)g+1}; \ldots; \mathbf{m}_{qg}]\right)\right), \qquad \overline{\mathbf{m}}_q = \sum_{\ell=1}^{g} \alpha_{q,\ell} \mathbf{m}_{(q-1)g+\ell}. \tag{1}$$

Here, $s \in \{\mathrm{prompt}, \mathrm{inference}\}$ identifies the memory type, and $f_\phi^{(s,q)} : \mathbb{R}^{g \times d_W} \to \mathbb{R}^g$ denotes the scoring function for group q of compressor s, with ϕ collecting all compressor parameters. Softmax normalizes the scores within each group. Applying this operation to all groups produces a compressed sequence $\bar{M} = [\bar{\mathbf{m}}_1, \ldots, \bar{\mathbf{m}}_{K/g}]$, reducing sequence length while preserving the hidden dimension. Architecture and dataset-specific compression settings are provided in Appendix A.8.

Both compressors are jointly optimized with the Trigger, while the Weaver, Reasoner, and cross-model projections remain frozen. The joint objective is

$$\mathcal{L}_{\mathrm{CSMC}} = \mathcal{L}_{\mathrm{trig}} + \mathcal{L}_{\mathrm{comp}}, \tag{2}$$

with equal loss weights ($\lambda = 1$). The compressor loss applies a clipped likelihood-ratio surrogate to answer tokens conditioned on compressed memory, using the same group-relative terminal-reward advantage as the Trigger. During trajectory replay, sampled tokens and Trigger decisions are held fixed, while compressed memories are recomputed. Gradients propagate through the frozen Reasoner to both compressors. Appendix A.8 provides the objective and optimization details.

3.2 Depth-Aware Readout (DAR)

Motivation. In single-loop continuous-thought architectures, latent memory is conventionally extracted from the final transformer layer. However, mechanistic interpretability studies suggest that late-layer representations become increasingly aligned with vocabulary-space predictions, whereas intermediate layers encode richer abstract and conceptual task representations (Meng et al., 2022; Hendel et al., 2023; Todd et al., 2024). This distinction is particularly relevant to latent memory, whose representations are re-injected into the Reasoner's hidden stream rather than directly decoded into tokens. Consequently, the layer best suited for immediate token prediction may not be optimal for supporting subsequent reasoning, motivating a systematic investigation of intermediate-layer readout.

Depth-aware extraction. Motivated by this functional misalignment, we introduce Depth-Aware Readout (DAR), which formalizes extraction depth as a configurable design variable. Let the Weaver be an L-layer transformer operating on a sequence of length $T = T_{\text{ctx}} + K$, formed by concatenating T_{ctx} contextual embeddings (mapped from the reasoner) with K learnable query-latent slots appended at the tail; let $\mathbf{H}^{(\ell)} \in \mathbb{R}^{T \times d_W}$ denote the sequence of hidden states at layer $\ell \in \{0, 1, \ldots, L\}$, where $\ell{=}0$ corresponds to the input embedding and $\ell{=}L$ to the final transformer block. While the baseline rigidly restricts extraction to the terminal layer ($\ell^\star = L$), DAR generalizes this to a configurable depth $\ell^\star \in \{1, \ldots, L\}$. Specifically, the fixed-length latent memory tokens are read out from the K trailing query-latent positions at this designated layer, which have absorbed contextual information through self-attention:

$$\mathbf{M}_t = \mathbf{H}^{(\ell^\star)}_{[T-K:T,\,:]} \in \mathbb{R}^{K \times d_W}, \tag{3}$$

Before injection, each memory vector is projected into the Reasoner's hidden space: $\mathbf{M}_t^R = \mathbf{M}_t \mathbf{W}_{W \to R}^\top \in \mathbb{R}^{K \times d_R}$, where $\mathbf{W}_{W \to R} \in \mathbb{R}^{d_R \times d_W}$ and d_R is the Reasoner's hidden dimension. Setting $\ell^\star = L$ naturally recovers the baseline, whereas setting $\ell^\star$ to a strictly intermediate value defines DAR. We operationalize DAR using a fixed intermediate depth in this work, leaving adaptive or task-conditional depth selection as a natural extension for future exploration.

3.3 REASONING-STATE-AWARE TRIGGER (RSAT)

Motivation. The third design choice we revisit is when latent memory should be invoked. The framework we build upon (Zhang et al., 2025) consults the Trigger at punctuation boundaries (periods, commas, and line breaks). Punctuation provides a simple structural cue for selecting candidate invocation positions, but does not explicitly account for the Reasoner's predictive uncertainty. A simple arithmetic solution may contain multiple punctuation marks even when the model proceeds confidently, whereas a difficult inferential step may occur within a clause without any boundary marker—precisely where additional reasoning support could be most valuable.

A more natural intuition comes from human reasoning: when we become uncertain about how to proceed, recalling a relevant fact, principle, or worked example can be especially useful. Modern LLMs provide a model-intrinsic analogue through predictive entropy. At each decoding step, the next-token distribution defines H_t, providing a local measure of predictive uncertainty. Unlike punctuation, H_t is available at every step and reflects the model's current predictive state. Predictive uncertainty has previously been used to gate *external* retrieval mid-generation (Jiang et al., 2023; Su et al., 2024) and, in concurrent work, to switch a single model between explicit and latent decoding modes (Shi et al., 2025); we adapt the same uncertainty-driven principle to control the injection of an internal, learned latent-memory module. Appendix D provides a training-free empirical analysis showing that H_t is statistically associated with reasoning phase, phase transitions, and problem difficulty.

Predictive entropy. Let $p_t(\cdot \mid x, y_{<t}) \in \Delta^{|\mathcal{V}|-1}$ denote the Reasoner's next-token distribution over vocabulary $\mathcal{V}$ at decoding step t. For brevity, this notation suppresses the latent-memory context already incorporated into the decoding state. We define the Reasoner's *instantaneous predictive entropy* as

$$H_t \;=\; H(p_t(\cdot \mid x, y_{<t})) \;=\; -\sum_{v \in \mathcal{V}} p_t(v \mid x, y_{<t}) \log p_t(v \mid x, y_{<t}). \tag{4}$$

H_t is small when the reasoner is confident about the next token (the distribution is peaked) and large when it is uncertain (the distribution is spread). This per-step signal is computed as a by-product of standard autoregressive decoding and is therefore freely available at every position, not only at delimiters.

Entropy-gated triggering. We propose the **Reasoning-State-Aware Trigger (RSAT)**: instead of consulting the Trigger only at punctuation boundaries, RSAT consults the Trigger at positions where the reasoner's predictive entropy lies in an uncertainty band and a minimum-distance cooldown since the last memory injection has elapsed. Concretely, RSAT replaces the punctuation-gated decision rule with

$$d_t = \begin{cases} \pi_{\text{trig}}(x, y_{<t}; \theta_T), & \text{if } H_t \in [\tau_{\text{low}}, \tau_{\text{high}}) \text{ and } t - t_{\text{last}} \geq D, \\ \text{SKIP}, & \text{otherwise,} \end{cases} \tag{5}$$

where π_{trig} is the same LoRA-parameterized Trigger used in the baseline (with parameters θ_T), t_{last} is the most recent decoding step at which the memory was injected (initialized to $-\infty$), and $d_t \in \{\text{INVOKE}, \text{SKIP}\}$ is the binary decision over whether to fire the Weaver at step t. Three remarks justify this form. First, the *lower threshold* τ_{low} is the operative knob—it implements the core principle of consulting the Trigger only when the reasoner is genuinely uncertain. Second, the upper threshold τ_{high} bounds the candidate uncertainty range, excluding the extreme upper tail of the entropy distribution. We fix this threshold at P_{95} and vary the lower threshold to compare different uncertainty bands. Third, the *cooldown* D prevents the fixed per-generation Trigger budget (here, at most five memory injections per generation; see Appendix A.3) from being exhausted inside a single high-entropy cluster, which would leave later, equally informative high-entropy points unattended. Setting $\tau_{\text{high}} \to \infty$ and $D = 0$ recovers the single-threshold rule $H_t \geq \tau_{\text{low}}$; we adopt the band-plus-cooldown form as the default operationalization, and report the relevant hyperparameters $(\tau_{\text{low}}, \tau_{\text{high}}, D)$ in Section 4.4.

Let $\mathcal{P}$ denote the set of delimiter tokens, such as commas, periods, and line breaks. The baseline selects candidate invocation positions using $\mathbf{1}[y_{t-1} \in \mathcal{P}]$, whereas RSAT uses $\mathbf{1}[H_t \in [\tau_{\text{low}}, \tau_{\text{high}})] \cdot \mathbf{1}[t - t_{\text{last}} \geq D]$. At eligible positions, the learned Trigger determines whether memory is actually injected. The first factor selects steps whose predictive entropy, computed from the full next-token distribution, falls within the specified interval, while the second enforces a cooldown of D steps

between successive memory injections. These two criteria need not coincide: punctuation may occur at either low- or high-uncertainty positions, whereas elevated uncertainty may arise within a clause without any nearby delimiter. RSAT therefore replaces a surface-token heuristic with a model-intrinsic uncertainty signal.

4 EXPERIMENTS

We evaluate the three modifications introduced in Section 3 along the same What–Where–When axes. Section 4.1 describes the common experimental setup, and Sections 4.2–4.4 report results for CSMC, DAR, and RSAT respectively. Throughout, we hold the dual-module Weaver–Trigger architecture and all non-targeted hyperparameters strictly fixed, so that any change in performance can be attributed to the specific design variable under study.

4.1 EXPERIMENTAL SETUP

Base reasoners. We instantiate the framework with four frozen reasoners spanning different model generations and scales: Qwen2.5-7B-Instruct (Yang et al., 2024a), Qwen3-4B-Instruct (Yang et al., 2025), and Qwen3.5-2B and Qwen3.5-4B (Qwen Team, 2026). The Reasoner parameters are never updated. The Weaver is first trained through supervised fine-tuning. During the subsequent reinforcement-learning stage, the Reasoner and Weaver remain frozen, while the Trigger is optimized in all configurations; for CSMC, the two compressors are jointly optimized with the Trigger.

Datasets. We evaluate on four reasoning benchmarks covering distinct capability profiles: GSM8K (Cobbe et al., 2021) for grade-school multi-step arithmetic, GPQA (Rein et al., 2023) for graduate-level scientific reasoning, SVAMP (Patel et al., 2021) for structurally varied mathematical word problems, and StrategyQA (Geva et al., 2021) for multi-hop commonsense reasoning. CSMC uses teacher-generated CoT targets only during Weaver training, while DAR and RSAT use the dataset's native supervision.

Baseline. We construct a matched baseline for each of the three design axes. For CSMC (Section 3.1), we compare three configurations. **Native MemGen** follows the original MemGen framework (Zhang et al., 2025), using the dataset's native concise rationales and uncompressed latent memory. **CoT (Uncompressed)** replaces only the supervision target with teacher-generated CoT while retaining the full candidate memory. **CSMC (Compressed)** uses the same CoT training data and the same CoT-supervised Weaver as the uncompressed configuration, but applies learned compression before memory injection. Native MemGen versus CoT (Uncompressed) isolates the effect of richer reasoning supervision, whereas CoT (Uncompressed) versus CSMC (Compressed) isolates the effect of memory compression. CoT (Uncompressed) therefore serves as the direct baseline for evaluating CSMC.

For DAR (Section 3.2), final-layer readout ($\ell^\star = L$) serves as the direct baseline for intermediate-layer readout ($\ell^\star = \lfloor L/2 \rfloor$); both configurations use native supervision and punctuation-gated triggering. For RSAT (Section 3.3), the punctuation-gated Trigger serves as the direct baseline for entropy-gated triggering; both configurations use native supervision and final-layer readout. Full training and evaluation hyperparameters are provided in Appendix A.

4.2 WHAT TO REMEMBER: CSMC

Setup. We evaluate the three configurations defined in Section 4.1: Native MemGen, CoT (Uncompressed), and CSMC (Compressed). The two CoT-based configurations use the same training data and the same CoT-supervised Weaver checkpoint. For CSMC, both compressors are jointly optimized with the Trigger before the compressed memory is injected into the Reasoner. All configurations use final-layer Weaver readout, punctuation-gated triggering, and the same evaluation and inference settings. Detailed compression and training configurations are provided in Appendix A.

Main result. Table 1 separates the effects of CoT supervision and memory compression through two pairwise comparisons. Replacing native supervision with CoT supervision improves accuracy in 15 of the 16 backbone–dataset configurations, with an average gain of 9.30 percentage points.

Table 1: CSMC results (accuracy, %).

Dataset	Native MemGen	CoT (uncompressed)	CSMC (compressed)
Qwen2.5-7B-Instruct			
GSM8K	73.77	89.69 (+15.92)	89.76 (+0.07)
GPQA	21.21	33.84 (+12.63)	33.33
SVAMP	82.33	91.67 (+9.34)	92.00 (+0.33)
StrategyQA	63.17	73.07 (+9.90)	72.20
Qwen3-4B-Instruct			
GSM8K	80.89	89.54 (+8.65)	89.69 (+0.15)
GPQA	18.69	33.84 (+15.15)	34.85 (+1.01)
SVAMP	76.67	90.67 (+14.00)	91.33 (+0.66)
StrategyQA	71.76	75.84 (+4.08)	66.96
Qwen3.5-2B			
GSM8K	66.72	68.84 (+2.12)	68.76
GPQA	7.58	16.67 (+9.09)	14.14
SVAMP	71.33	73.67 (+2.34)	78.00 (+4.33)
StrategyQA	49.93	49.78	51.97 (+2.19)
Qwen3.5-4B			
GSM8K	80.14	87.49 (+7.35)	89.46 (+1.97)
GPQA	18.69	39.90 (+21.21)	32.83
SVAMP	73.33	87.67 (+14.34)	79.00
StrategyQA	62.30	65.07 (+2.77)	58.52

Green subscripts: positive gains (pp); CoT vs. Native, CSMC vs. CoT.

The largest improvement occurs on Qwen3.5-4B GPQA, where accuracy increases from 18.69% to 39.90%. These results demonstrate the value of richer reasoning targets for training the Weaver.

Relative to uncompressed CoT memory, CSMC improves accuracy in eight configurations while injecting fewer memory vectors, with gains of 4.33 percentage points on Qwen3.5-2B SVAMP and 2.19 points on Qwen3.5-2B StrategyQA. On Qwen2.5-7B, accuracy remains within one percentage point of the uncompressed configuration on all four benchmarks. The complete CSMC system also outperforms Native MemGen in 14 of the 16 configurations. Together, these results show that CoT supervision broadly improves performance and that compact memories can retain or improve upon uncompressed performance in multiple settings, with the effect of compression varying across backbones and tasks.

4.3 WHERE TO EXTRACT: DAR

Setup. We compare final-layer readout ($\ell^\star = L$) with a fixed intermediate-layer readout ($\ell^\star = \lfloor L/2 \rfloor$) across all four backbones and datasets. Both configurations use native Weaver supervision and the same punctuation-gated Trigger; only the extraction depth is changed.

Main result. Table 2 shows that intermediate-layer readout outperforms last-layer readout in 11 of the 16 backbone–dataset configurations and matches it in one, increasing average accuracy from 57.41% to 61.91%. On GPQA, it improves three backbones and matches the remaining one, with gains of 6.56, 10.60, and 12.62 percentage points on Qwen3-4B, Qwen3.5-2B, and Qwen3.5-4B, respectively. Qwen3.5-4B benefits across all four datasets, including a 15.67-point gain on SVAMP. Although four configurations favor last-layer readout, the improvements across most evaluated settings demonstrate that intermediate layers are an effective source of latent memory and that readout depth is a consequential design choice.

Table 2: DAR results (accuracy, %).

Model	Dataset	Final-layer	Middle-layer	Gain (pp)
Qwen2.5-7B-Instruct	GSM8K	73.77	73.09	–
	GPQA	21.21	21.21	–
	SVAMP	82.33	87.67	↑5.34
	StrategyQA	63.17	65.07	↑1.90
Qwen3-4B-Instruct	GSM8K	80.89	80.74	–
	GPQA	18.69	25.25	↑6.56
	SVAMP	76.67	89.67	↑13.00
	StrategyQA	71.76	68.70	–
Qwen3.5-2B	GSM8K	66.72	68.01	↑1.29
	GPQA	7.58	18.18	↑10.60
	SVAMP	71.33	63.00	–
	StrategyQA	49.93	55.31	↑5.38
Qwen3.5-4B	GSM8K	80.14	82.11	↑1.97
	GPQA	18.69	31.31	↑12.62
	SVAMP	73.33	89.00	↑15.67
	StrategyQA	62.30	72.20	↑9.90

Green arrows indicate gains over final-layer readout (pp).

4.4 When to recall: RSAT

Setup. We compare RSAT with the punctuation-gated baseline defined in Section 4.1. RSAT uses entropy-gated triggering, with τ_{low} and τ_{high} calibrated as percentiles of the Reasoner's predictive-entropy distribution on the training prompts. We evaluate four entropy bands by sweeping $\tau_{\text{low}} \in \{P_{75}, P_{80}, P_{85}, P_{90}\}$ while fixing $\tau_{\text{high}} = P_{95}$ and the cooldown distance to $D = 16$ tokens. All configurations use native supervision and final-layer Weaver readout, with the remaining training and inference settings held fixed.

To identify a robust default band across backbones and tasks, we aggregate the within-configuration rankings of the candidate bands. For each backbone–dataset configuration i, the B candidate bands are ranked by accuracy, with tied results assigned their average rank. Let r_{ij} denote the rank of band I_j in configuration i. We define its normalized rank score as

$$S(I_j) = \frac{100}{N(B-1)} \sum_{i=1}^{N} (B - r_{ij}), \qquad I^* = \arg\max_{I_j} S(I_j). \tag{6}$$

where N is the number of evaluated backbone–dataset configurations. All configurations receive equal weight. Maximizing $S(I_j)$ is equivalent to minimizing its average rank. The punctuation-gated baseline is excluded from the ranking and is used only to measure the absolute benefit of entropy-gated triggering.

Main result. Table 3 compares RSAT across four entropy bands against punctuation-gated triggering. The $[P_{80}, P_{95})$ band achieves the best aggregate ranking across the 16 backbone–dataset configurations, with an average rank of 2.31 and a Borda score of 56.25. It is closely followed by $[P_{85}, P_{95})$, whose average rank is 2.34. This comparison identifies $[P_{80}, P_{95})$ as the strongest aggregate choice among the evaluated bands.

Relative to punctuation-gated triggering, $[P_{80}, P_{95})$ improves accuracy in nine configurations and matches it in one, with an average gain of 2.05 percentage points. Notable improvements include 12.96 points on Qwen2.5-7B StrategyQA, 9.25 points on Qwen3.5-4B GSM8K, and 8.33 points on Qwen3-4B SVAMP. These results support predictive uncertainty as a useful criterion for selecting candidate memory-invocation positions, while the variation across bands highlights the importance of the operating range.

Table 3: RSAT results (accuracy, %).

Model	Dataset	Punctuation	$[P_{75}, P_{95})$	$[P_{80}, P_{95})$	$[P_{85}, P_{95})$	$[P_{90}, P_{95})$
Qwen2.5-7B	GSM8K	73.77	72.48	72.63	72.33	**76.04**
	GPQA	21.21	23.23	26.26 (+5.05)	**26.77**	23.23
	SVAMP	82.33	81.33	82.33	**83.33**	82.33
	StrategyQA	63.17	72.63	**76.13** (+12.96)	73.80	75.11
Qwen3-4B	GSM8K	80.89	80.59	**80.67**	**80.67**	80.52
	GPQA	18.69	**18.69**	17.68	17.17	14.65
	SVAMP	76.67	84.00	**85.00** (+8.33)	82.33	83.00
	StrategyQA	71.76	59.24	67.39	66.08	**73.22**
Qwen3.5-2B	GSM8K	66.72	61.79	58.98	62.62	**65.66**
	GPQA	7.58	8.59	8.08 (+0.50)	**10.61**	6.57
	SVAMP	71.33	60.67	66.67	77.33	**78.33**
	StrategyQA	49.93	53.71	51.09 (+1.16)	48.18	**57.06**
Qwen3.5-4B	GSM8K	80.14	88.25	**89.39** (+9.25)	88.78	87.87
	GPQA	18.69	20.71	20.20 (+1.51)	**23.23**	20.71
	SVAMP	73.33	**83.00**	81.00 (+7.67)	68.67	64.33
	StrategyQA	62.30	67.39	67.83 (+5.53)	68.85	**69.58**
	Average rank ↓		2.88	**2.31**	2.34	2.47
	Borda score ↑		37.50	**56.25**	55.21	51.04

Bold/underline: best/second-best bands per row. Green subscripts: gains over baseline (pp). Ranks use all 16 configurations, with ties averaged; baseline excluded. Gray: best aggregate.

5 Discussion

CSMC connects richer CoT supervision with task-driven memory compression. The pairwise comparisons show that supervision quality and memory capacity play distinct roles: CoT targets strengthen the Weaver, while the learned compressor controls how that information is conveyed through fewer latent vectors. The ability to preserve or improve accuracy after compression in several settings suggests that injecting every candidate vector is not always necessary, while variation across configurations motivates adapting memory capacity to the task.

DAR and RSAT address how the resulting memory is accessed and used. DAR extracts intermediate-layer representations, whose advantage in most evaluated settings shows that the final layer is not always the most effective source for subsequent injection. RSAT uses predictive entropy to select candidate invocation positions, allowing recall decisions beyond punctuation boundaries. Together, these findings suggest that effective latent memory depends on matching its content, representation, and timing to the Reasoner. Jointly adapting these choices is a promising extension of the separate comparisons studied here.

6 Conclusion

We revisit latent memory generation through the What–Where–When perspective. Across four Qwen backbones and four reasoning benchmarks, CoT supervision improves Weaver training, intermediate-layer readout outperforms last-layer readout in most evaluated configurations, and entropy-gated triggering improves aggregate performance over punctuation-based candidate selection. CSMC further explores the trade-off between memory capacity and accuracy, yielding more compact memories with configuration-dependent effects on performance. Together, these findings establish memory content and capacity, extraction depth, and recall timing as consequential design choices for augmenting frozen reasoners, and motivate their joint adaptation.

7 AI USE STATEMENT

We used generative AI tools to assist with literature search and discovery, code development and revision, and manuscript drafting and editing. We used GPT-4o to generate chain-of-thought explanations for Weaver training, as described in Appendix C, while retaining the original dataset questions and reference answers. We also used an LLM-based assistant to help organize the discussion and improve the clarity and readability of the manuscript. The authors reviewed the AI-assisted text and code, verified the cited sources and numerical comparisons, and take full responsibility for the final manuscript and all associated research artifacts.

REFERENCES

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

Eunseong Choi, June Park, Hyeri Lee, and Jongwuk Lee. Conflict-aware soft prompting for retrieval-augmented generation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025.

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. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Yuntian Deng, Kiran Prasad, Roland Fernandez, Paul Smolensky, Vishrav Chaudhary, and Stuart Shieber. Implicit chain of thought reasoning via knowledge distillation. *arXiv preprint arXiv:2311.01460*, 2023.

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

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics (TACL)*, 9:346–361, 2021.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

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

Roee Hendel, Mor Geva, and Amir Globerson. In-context learning creates task vectors. In *Findings of the Association for Computational Linguistics: EMNLP*, 2023.

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

Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Mingyu Jin, Qinkai Yu, Jingyuan Huang, Qingcheng Zeng, Zhenting Wang, Wenyue Hua, Haiyan Zhao, Kai Mei, Yanda Meng, Kaize Ding, Fan Yang, Mengnan Du, and Yongfeng Zhang. Exploring concept depth: How large language models acquire knowledge and concept at different layers? In *Proceedings of the 31st International Conference on Computational Linguistics (COLING)*, 2025.

Luyang Liu, Jonas Pfeiffer, Jiaxing Wu, Jun Xie, and Arthur Szlam. Deliberation in latent space via differentiable cache augmentation. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

Tianqiao Liu, Zui Chen, Zitao Liu, Mi Tian, and Weiqi Luo. Expediting and elevating large language model reasoning via hidden chain-of-thought decoding. *arXiv preprint arXiv:2409.08561*, 2024.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

nostalgebraist. Interpreting GPT: The logit lens. LessWrong, 2020. URL https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 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 (NAACL-HLT)*, pp. 2080–2094, 2021.

Qwen Team. Qwen3.5. https://huggingface.co/collections/Qwen/qwen35, 2026.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof Q&A benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

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

Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. CODI: Compressing chain-of-thought into continuous space via self-distillation. *arXiv preprint arXiv:2502.21074*, 2025.

Dachuan Shi, Abedelkadir Asi, Keying Li, Xiangchi Yuan, Leyan Pan, Wenke Lee, and Wen Xiao. SwiReasoning: Switch-thinking in latent and explicit for pareto-superior reasoning LLMs. *arXiv preprint arXiv:2510.05069*, 2025.

Oscar Skean, Md Rifat Arefin, Dan Zhao, Niket Patel, Jalal Naghiyev, Yann LeCun, and Ravid Shwartz-Ziv. Layer by layer: Uncovering hidden representations in language models. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

Weihang Su, Yichen Tang, Qingyao Ai, Zhijing Wu, and Yiqun Liu. DRAGIN: Dynamic retrieval augmented generation based on the real-time information needs of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Eric Todd, Millicent L. Li, Arnab Sen Sharma, Aaron Mueller, Byron C. Wallace, and David Bau. Function vectors in large language models. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

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

Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. In *The Tenth International Conference on Learning Representations (ICLR)*, 2022.

Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. SoftCoT: Soft chain-of-thought for efficient reasoning with LLMs. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024a.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Sohee Yang, Elena Gribovskaya, Nora Kassner, Mor Geva, and Sebastian Riedel. Do large language models latently perform multi-hop reasoning? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024b.

Zijun Yao, Weijian Qi, Liangming Pan, Shulin Cao, Linmei Hu, Weichuan Liu, Lei Hou, and Juanzi Li. SeaKR: Self-aware knowledge retrieval for adaptive retrieval-augmented generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025.

Guibin Zhang, Muxin Fu, and Shuicheng Yan. MemGen: Weaving generative latent memory for self-evolving agents. *arXiv preprint arXiv:2509.24704*, 2025.

A Implementation Details

A.1 Models and Datasets

We evaluate MemGen on four backbone models: Qwen2.5-7B-Instruct, Qwen3-4B-Instruct, Qwen3.5-2B, and Qwen3.5-4B. The Weaver and Trigger share the same base architecture as the Reasoner and are adapted via LoRA (Hu et al., 2022).

Training uses the supervision data corresponding to each configuration, with a 90/10 training/validation split. For the CoT-based configurations, supervision targets combine teacher-generated explanations with the original reference answers, as described in Appendix C. The default random seed is 42.

A.2 LoRA Configuration

Both the Weaver and Trigger are fine-tuned with LoRA applied to the query- and value-projection matrices. The same configuration is used for both components and all four backbone models (Table 4).

Table 4: LoRA hyperparameters (shared by Weaver and Trigger).

Hyperparameter	**Value**
Rank r	16
Alpha α	32
Dropout	0.1
Target modules	`q_proj`, `v_proj`
Bias	none

A.3 Latent Memory Architecture

Latent memory tokens are generated by the Weaver and injected into the Reasoner's KV cache at selected positions. Table 5 summarizes the architectural hyperparameters.

A.4 Weaver Training

The Weaver is trained with supervised fine-tuning (SFT). Hyperparameters are listed in Table 6.

Table 5: Latent memory architectural hyperparameters.

Hyperparameter	Value
Prompt latent length (L_p)	16 tokens
Inference latent length (L_i)	8 tokens
Max prompt augmentation count	1
Max inference augmentation count N	5

Table 6: Weaver SFT training hyperparameters.

Hyperparameter	Value
Training epochs	2
Per-device batch size	1
Gradient accumulation steps	1
Optimizer	AdamW
Learning rate	1×10^{-5}
LR scheduler	Cosine
Warmup ratio	0.1

A.5 Trigger Training

The Trigger is trained with Group Relative Policy Optimisation (GRPO) (Shao et al., 2024) using the BNPO loss variant. The Weaver checkpoint is loaded and frozen during Trigger training. Hyperparameters are listed in Table 7.

Table 7: Trigger GRPO training hyperparameters.

Hyperparameter	Value
Training epochs	1
Per-device batch size	8
Gradient accumulation steps	4
Effective batch size	32
Number of generations per prompt (G)	8
Loss type	BNPO
KL penalty β	0.0
Optimizer	AdamW
Learning rate	1×10^{-5}
LR scheduler	Cosine
Warmup ratio	0.1

A.6 Reasoning-State-Aware Trigger

For RSAT, candidate memory-injection positions are selected based on token-level predictive entropy rather than delimiter positions. Entropy thresholds are derived from the entropy distribution computed over the training set. Table 8 lists the relevant hyperparameters.

We vary the lower cutoff over $\{P_{75}, P_{80}, P_{85}, P_{90}\}$ while fixing the upper cutoff at P_{95}. Lower cutoffs admit a broader range of candidate positions, whereas higher cutoffs restrict eligibility to positions with higher predictive entropy. Table 3 reports the results for all four bands. Although the best-performing band varies across backbones and datasets, $[P_{80}, P_{95})$ achieves the best aggregate ranking across the evaluated configurations and is adopted as our default. A minimum distance of $D = 16$ tokens between successive memory injections enforces the cooldown.

Table 8: Reasoning-State-Aware Trigger hyperparameters.

Hyperparameter	**Value**
Entropy lower-bound percentile τ_{low}	$\{P75, P80, P85, P90\}$ (swept)
Entropy upper-bound percentile	P95
Minimum insertion distance	16 tokens

Table 9: Evaluation hyperparameters and protocol.

Hyperparameter / Protocol	Value
Default random seed	42
Runs per configuration	1
Answers per question	1
Majority voting	None
Temperature	1.0 for sampling; N/A for greedy
Repetition penalty	1.0
Stopping criterion	First EOS or generation-length limit
Accuracy aggregation	Mean correctness over questions
Macro-average accuracy	Equal weight per backbone–dataset pair
GSM8K evaluation questions	1,319
GPQA evaluation questions	198
SVAMP evaluation questions	300
StrategyQA evaluation questions	687

A.7 Evaluation Settings

Table 9 summarizes the evaluation hyperparameters and protocol.

A.8 Memory Compression Configuration

CSMC uses two independently parameterized compressors for prompt memory and inference memory. Both preserve the backbone hidden dimension d. Within each compressor, each input vector is processed by LayerNorm, a linear projection from d to 64 dimensions, and a Tanh activation. For a group of g vectors, the resulting features are concatenated into a $64g$-dimensional representation. Each group has its own linear scoring head, $\mathrm{Linear}(64g, g)$, followed by a softmax over the g scores. The compressed vector is the weighted sum of the original d-dimensional vectors in that group.

We use $g = 2$ for GPQA and $g = 4$ for GSM8K, SVAMP, and StrategyQA. These group sizes were specified before evaluation and held fixed across backbones within each dataset. The candidate prompt and inference memories contain 16 and 8 vectors, respectively. Compression therefore produces 8 and 4 vectors for GPQA, and 4 and 2 vectors for the other datasets. All scoring-head weights and biases are initialized to zero, yielding uniform initial weights of $1/g$ and making the initial compression equivalent to within-group mean pooling.

For each input x, we sample $G = 8$ complete trajectories under the old Trigger and compressor parameters. Each trajectory contains the Trigger decisions and the generated answer sequence. Let R_i denote the terminal task reward for trajectory i. We use the same group-relative advantage as the Trigger:

$$A_i = \mathrm{stopgrad}\left(\frac{R_i - \bar{R}}{\sigma_R + \delta}\right), \qquad \bar{R} = \frac{1}{G}\sum_{j=1}^{G} R_j,$$

where σ_R is the within-group reward standard deviation and $\delta > 0$ is a numerical stabilizer. The advantage is held fixed during optimization.

During replay, the sampled answer tokens and Trigger decisions are held fixed, while compressed memories are recomputed using the current compressor parameters ϕ. Let $\mathcal{S}_i$ denote the answer-token positions conditioned on at least one compressed-memory insertion, including prompt-

Table 10: Memory compression configurations.

Parameter	GPQA	GSM8K / SVAMP / StrategyQA
Input/output dimension	d	d
Hidden dimension	64	64
Group size g	2	4
Scoring head	$\mathrm{Linear}(128, 2)$	$\mathrm{Linear}(256, 4)$
Weight normalization	Softmax	Softmax
Prompt memory length	$16 \to 8$	$16 \to 4$
Inference memory length	$8 \to 4$	$8 \to 2$
Scoring-head initialization	Zero	Zero

memory insertion. For $t \in \mathcal{S}_i$, define

$$\rho_{i,t}^{Y}(\phi) = \frac{p_{\Theta,\phi}(y_{i,t} \mid x, y_{i,<t}, a_{i,\le t})}{p_{\Theta,\phi_{\mathrm{old}}}(y_{i,t} \mid x, y_{i,<t}, a_{i,\le t})},$$

where $a_{i,\le t}$ denotes the fixed Trigger-decision history before predicting $y_{i,t}$, and Θ collects the frozen Weaver, Reasoner, and cross-model projection parameters. The denominator is the detached probability under the old compressed-memory trajectory.

The compressor surrogate is

$$\mathcal{L}_{\mathrm{comp}} = -\mathbb{E}_{i,\, t\in\mathcal{S}_i}\left[\min\left(\rho_{i,t}^{Y}(\phi)A_i,\ \mathrm{clip}\big(\rho_{i,t}^{Y}(\phi), 1-\epsilon, 1+\epsilon\big)A_i\right)\right],$$

where ϵ is the clipping parameter. Trajectories with $\mathcal{S}_i = \varnothing$ contribute no compressor-loss terms. No auxiliary reconstruction loss or separate supervised compressor objective is used.

Freezing model parameters does not disable differentiation with respect to their inputs. Answer-token likelihood gradients propagate through the frozen Reasoner and memory projections to the compressors. When a later Weaver invocation depends on earlier compressed memories, the corresponding dependency is also retained during replay. Thus, the terminal reward weights the likelihood gradients through A_i; the reward itself is not differentiated.

The compressor surrogate is normalized by the number of affected answer tokens. During backpropagation, parameter-gradient contributions from prompt and inference compression are balanced separately using the corresponding per-trajectory invocation counts and the numbers of active trajectories for each role. Both role-specific gradient weights are set to 1. These gradient-balancing weights are distinct from the unit coefficients of the two loss terms in Equation 2.

B Related Work

Latent reasoning. A growing line of work explores latent reasoning, representing intermediate reasoning steps as continuous hidden states rather than discrete tokens. Methods such as Coconut (Hao et al., 2024) feed hidden states back as input embeddings; Implicit CoT-KD (Deng et al., 2023) distills explicit teacher reasoning into student hidden states, while Stepwise Internalization (Deng et al., 2024) achieves a similar effect through a curriculum that gradually removes CoT tokens during finetuning. CCoT (Cheng & Van Durme, 2024), CODI (Shen et al., 2025), and SoftCoT (Xu et al., 2025) encode explicit reasoning into continuous representations through compression, self-distillation, or auxiliary soft thoughts. Unlike these approaches, CSMC first uses richer CoT supervision to provide candidate reasoning information and then learns which information to retain under a reduced latent-memory budget. More recently, SIM-CoT (Wei et al., 2025) identifies a *latent-collapse* failure mode in implicit-CoT methods and resolves it through step-level supervision that aligns each latent token to its corresponding explicit reasoning step. Our work belongs to the family of *single-loop continuous-thought architectures*, in which an auxiliary module produces latent tokens that are fed back to the main reasoner during generation. HCoT (Liu et al., 2024) is a close architectural analogue to the LMG paradigm we adopt: it couples a frozen reasoner with an auxiliary model that emits a single compressed CoT embedding for use during decoding. Across related methods, the positions at which latent reasoning or memory can be used are often

constrained by fixed slots (Hao et al., 2024; Xu et al., 2025; Shen et al., 2025), punctuation-based eligibility rules (Zhang et al., 2025), or fixed prefixes (Goyal et al., 2024). SwiReasoning (Shi et al., 2025) also uses predictive entropy as a control signal, but switches a single model between explicit and latent decoding modes rather than controlling access to an externally generated latent-memory module. Our RSAT directly targets this underexamined design variable.

Layer selection in transformer representations. Mechanistic interpretability reveals that transformer layers serve distinct functional roles: factual associations are stored primarily in middle-layer MLPs (Meng et al., 2022), in-context task representations concentrate in intermediate layers (Hendel et al., 2023; Todd et al., 2024), while the residual stream of late layers progressively narrows toward vocabulary-aligned token prediction (nostalgebraist, 2020). Recent layer-wise probing studies sharpen this picture in two complementary ways: across diverse downstream tasks, intermediate layers consistently outperform final-layer embeddings, sometimes by double-digit margins (Skean et al., 2025); and within reasoning tasks specifically, layer-wise concept-probe accuracy follows an inverted-U over depth, peaking at intermediate layers while final layers are dominated by next-token features (Jin et al., 2025). Causal evidence from multi-hop reasoning further shows that bridge entities and partial inferences materialize at specific intermediate depths before being projected into next-token logits (Yang et al., 2024b). Building on these observations, several recent works have begun to ablate which layer's hidden states should be extracted: CCoT (Cheng & Van Durme, 2024) reports that mid-layer extraction yields the best performance, with extraction near the final layer failing to converge; Implicit CoT-KD (Deng et al., 2023) extracts across all layers, distilling from teacher to student layer-by-layer; and Differentiable Cache Augmentation (Liu et al., 2025) shows that compressing to a single final layer degrades performance relative to using the full multi-layer KV-cache. These findings suggest that extraction depth can materially affect performance and motivate evaluating intermediate-layer readout in LMG.

Memory-augmented LLMs. Augmenting LLMs with memory has emerged as a broad strategy for extending their effective context and supporting long-horizon reasoning. Memorizing Transformers (Wu et al., 2022) extend attention with a non-differentiable kNN lookup over external key-value pairs, while MemGPT (Packer et al., 2023) treats LLMs as operating systems with hierarchical context management. MemGen (Zhang et al., 2025), our direct baseline, places latent memory generation under a Weaver–Trigger architecture that augments a frozen reasoner with on-the-fly latent tokens; closely related, CARE (Choi et al., 2025) trains an auxiliary context-assessor that compresses retrieved context into learnable soft memory tokens consumed by a frozen base LLM, demonstrating that the "auxiliary module $\rightarrow$ soft memory tokens $\rightarrow$ frozen LLM" template extends naturally beyond reasoning to retrieval grounding. In parallel, predictive uncertainty has been successfully leveraged to gate external memory access: FLARE (Jiang et al., 2023) triggers retrieval mid-generation when token confidence drops, DRAGIN (Su et al., 2024) extends this with token-level next-token entropy, and SeaKR (Yao et al., 2025) further uses a self-aware uncertainty score extracted from the reasoner's internal feed-forward states to gate retrieval and re-rank passages. RSAT adapts this uncertainty-driven control principle inward: rather than triggering external retrieval, predictive entropy identifies candidate positions at which the Trigger may invoke an internal, learned latent-memory module.

C CoT Supervision Data Construction

C.1 Overview

CSMC uses teacher-generated step-by-step explanations as Stage-I supervision for the Weaver, while retaining the original questions and reference answers. Before constructing the supervised completions, we compare each teacher-generated final answer with the original reference answer using dataset-specific matching rules. Outputs with missing, unparseable, or mismatched final answers are regenerated and checked again. Only outputs that pass this check are used to construct CoT completions, which combine the generated rationale with the original reference answer in the required training format.

The CoT data shown below were generated with `gpt-4o`. The following system instruction is shared by all four datasets.

Common teacher instruction

```
Generate a clear and self-contained reasoning explanation for the
    given problem.

The explanation should:
- Directly address the specific problem.
- Include the intermediate reasoning or calculations needed to
    justify the answer.
- Remain logically consistent with the supplied answer when one is
    provided.
- Avoid generic problem-solving advice, irrelevant discussion, and
    unnecessary repetition.
- Follow the dataset-specific output format given below.
```

For each dataset, we describe the source fields and completion format, provide the dataset-specific instruction appended to the common instruction, and show matched native and CoT supervised completions. The teacher receives the question as the user message; SVAMP uses the concatenated problem statement and question, while GPQA additionally supplies the correct answer and distractors by role. Mathematical expressions in the reasoning are typeset for readability, while final-answer markers are shown literally to preserve the training format.

C.2 GSM8K

Source and target format. The source record contains a `question` and an `answer`. The reference answer is extracted from the original `answer` field after the `####` delimiter. The teacher generates a worked solution followed by the same delimiter and a final numerical answer. We compare the teacher's answer with the original reference answer after numerical normalization. Following the verification procedure described above, the builder combines the accepted rationale with the original reference answer, using a literal `\boxed{<answer>}` marker in the supervised completion.

GSM8K: teacher instruction

```
Solve the given mathematics word problem. Show the calculations and
    intermediate reasoning needed to obtain the answer.

OUTPUT FORMAT:
- End the response with a separate line in the form: ####
    Answer_Number
- After ####, output only the number. Do not include units, currency
    symbols, thousands separators, or additional text.
- Example: ...
#### 42
```

The following example compares native and CoT supervised completions for the same question. The CoT completion combines the accepted teacher-generated rationale with the original reference answer in the builder's final-answer format.

GSM8K: example question

Rosalina receives gifts from three people on her wedding day. How many gifts did she get if Emilio gave 11 gifts, Jorge gave 6 gifts, and Pedro gave 4 gifts?

GSM8K: native supervised completion

If Emilio presents 11 gifts and Jorge 6 gifts, together they have presented `11+6=«11+6=17»17` gifts.

If Pedro also gives Rosalinda 4 gifts, Rosalinda's total number is `17+4 = «17+4=21»21` gifts.

`\boxed{21}`

GSM8K: CoT supervised completion

To determine the total number of gifts Rosalina received on her wedding day, we need to add the number of gifts she received from each person. The problem states that Emilio gave her 11 gifts, Jorge gave her 6 gifts, and Pedro gave her 4 gifts.

To find the total, we perform the following calculation:

$$11 \text{ (gifts from Emilio)} + 6 \text{ (gifts from Jorge)} + 4 \text{ (gifts from Pedro)}.$$

Let's break this down step by step. First, add the gifts from Emilio and Jorge:

$$11 + 6 = 17.$$

Next, add the gifts from Pedro to this total:

$$17 + 4 = 21.$$

Thus, Rosalina received a total of 21 gifts from the three people combined.

To verify, let's revisit the steps. Adding 11 (Emilio's gifts) to 6 (Jorge's gifts) gives 17. Adding 4 (Pedro's gifts) to 17 yields 21. This confirms that the calculations are correct. Rosalina received 21 gifts in total.

This problem illustrates the basic principle of addition, where we combine quantities to find a total. Each step involves summing smaller groups of gifts to arrive at the final total. The solution is straightforward, as it only requires simple arithmetic addition and no complex operations.

In conclusion, the total number of gifts Rosalina received on her wedding day is 21, which is the sum of the gifts from Emilio, Jorge, and Pedro. The calculations are verified, and the answer is confirmed to be correct.

`\boxed{21}`

C.3 GPQA

Source and target format. Each source record contains `Question`, `Explanation`, `Correct Answer`, and three `Incorrect Answer` fields. The teacher receives the correct answer and distractors by role, without A–D letters, and generates an explanation followed by a separate final-answer line containing the selected option text. We check that this text identifies the original correct option. After verification, the final-answer line is removed and the explanation replaces `Explanation`. The builder shuffles the options and appends "Therefore, the final answer is" followed by the literal marker `\boxed{<letter>}`, with the letter determined from the original correct option under the shuffled order.

GPQA: teacher instruction

```
You will be given a multiple-choice question, the correct answer,
    and three distractor options.
Explain why the supplied correct answer follows from the relevant
    scientific principles. When useful, also clarify why the
    distractors are incorrect.

OUTPUT FORMAT:
- Provide the reasoning explanation.
- End with a separate line in the form:
  Final answer: <exact text of the correct option>
- Do not output an A/B/C/D option label.
- Do not use \boxed{...}.
```

The following example uses an illustrative option order. The final-answer letter appended by the builder corresponds to that order. Each supervised completion consists of the corresponding explanation followed by the builder-appended final answer.

GPQA: example question

Two quantum states with energies E_1 and E_2 have a lifetime of 10^{-9} sec and 10^{-8} sec, respectively. We want to clearly distinguish these two energy levels. Which one of the following options could be their energy difference so that they can be clearly resolved?

Illustrative option order

A. $10^{-4}\,\mathrm{eV}$
B. $10^{-11}\,\mathrm{eV}$
C. $10^{-8}\,\mathrm{eV}$
D. $10^{-9}\,\mathrm{eV}$

GPQA: native supervised completion

According to the uncertainty principle, $\Delta E \cdot \Delta t = \hbar/2$. Δt is the lifetime and ΔE is the width of the energy level. With $\Delta t = 10^{-9}\,\mathrm{s} \Longrightarrow \Delta E_1 = 3.3 \times 10^{-7}\,\mathrm{eV}$. And $\Delta t = 10^{-8}\,\mathrm{s}$ gives $\Delta E_2 = 3.3 \times 10^{-8}\,\mathrm{eV}$.

Therefore, the energy difference between the two states must be significantly greater than $10^{-7}\,\mathrm{eV}$. So the answer is $10^{-4}\,\mathrm{eV}$.

Therefore, the final answer is `\boxed{A}`

GPQA: CoT supervised completion

To clearly distinguish two quantum states with different energies, the energy difference between them must satisfy the criterion set by the uncertainty principle. The lifetime of each quantum state introduces an intrinsic uncertainty in energy, given by the energy-time uncertainty relation:

$$\Delta E \cdot \Delta t \geq \hbar.$$

where ΔE represents the uncertainty in energy, Δt is the lifetime of the state, and $\hbar$ is the reduced Planck constant ($\hbar \approx 6.58 \times 10^{-16}\,\mathrm{eV} \cdot \mathrm{s}$).

For a state with a lifetime of $\Delta t = 10^{-9}\,\mathrm{s}$, the energy uncertainty is:

$$\begin{aligned}\Delta E_1 &= \frac{\hbar}{\Delta t} \\ &= \frac{6.58 \times 10^{-16}\,\mathrm{eV} \cdot \mathrm{s}}{10^{-9}\,\mathrm{s}} \\ &= 6.58 \times 10^{-7}\,\mathrm{eV}.\end{aligned}$$

Similarly, for the state with $\Delta t = 10^{-8}\,\mathrm{s}$:

$$\begin{aligned}\Delta E_2 &= \frac{\hbar}{\Delta t} \\ &= \frac{6.58 \times 10^{-16}\,\mathrm{eV} \cdot \mathrm{s}}{10^{-8}\,\mathrm{s}} \\ &= 6.58 \times 10^{-8}\,\mathrm{eV}.\end{aligned}$$

To resolve the energy levels, the separation between their energies must exceed the combined uncertainties of the two states. The total uncertainty in energy is approximately:

$$\begin{aligned}\Delta E_{\text{total}} &= \Delta E_1 + \Delta E_2 \\ &\approx \left(6.58 \times 10^{-7} + 6.58 \times 10^{-8}\right)\mathrm{eV} \\ &\approx 7.24 \times 10^{-7}\,\mathrm{eV}.\end{aligned}$$

Thus, the energy difference between the two levels must be significantly larger than this value to ensure clear resolution. Among the options provided, only $10^{-4}\,\mathrm{eV}$ is sufficiently large to distinguish the levels, as it exceeds the uncertainty threshold by several orders of magnitude. The other options—$10^{-11}\,\mathrm{eV}$,

10^{-8} eV, and 10^{-9} eV—are much smaller than the combined uncertainty and would result in overlapping energy levels, making resolution impossible.

Therefore, the correct energy difference to clearly resolve the two states is 10^{-4} eV.

Therefore, the final answer is `\boxed{A}`

C.4 SVAMP

Source and target format. The source record contains `Body`, `Question`, their concatenation `question_concat`, `Equation`, the numeric `Answer`, and identifying metadata. The teacher receives `question_concat` and generates an explanation followed by a separate final numerical answer. We compare this answer with the original `Answer` after numerical normalization. After verification, the final-answer line is removed and the explanation replaces `Equation`; all other fields are preserved. The builder appends the original reference answer using the literal marker `\boxed{<answer>}`. Here, `Equation` is an inherited schema name; the replacement field contains the full explanation.

SVAMP: teacher instruction

```
Solve the given mathematics word problem. Show the calculations and
    intermediate reasoning needed to obtain the answer.

OUTPUT FORMAT:
- Provide the reasoning explanation.
- End with a separate line in the form:
  Final answer: <number>
- After "Final answer:", output only the number.
  Do not include units, currency symbols, thousands
  separators, or additional text.
- Do not use #### or \boxed{...}.
```

The following example retains the reference answer of 192. The teacher-generated explanation replaces the `Equation` field, and the builder appends the numeric final-answer marker to form the supervised completion.

SVAMP: example question

After Jessie started to go jogging everyday she lost 126 kilograms. She currently weighs 66 kilograms. How much did she weigh before starting to jog?

Reference answer

192

SVAMP: native supervised completion

$$(66.0 + 126.0)$$

`\boxed{192}`

SVAMP: CoT supervised completion

This problem asks us to determine Jessie's original weight before she started jogging, given that she lost 126 kilograms and now weighs 66 kilograms. To solve this, we need to think about how weight changes. When someone loses weight, their current weight is the result of subtracting the weight they lost from their starting weight. In this case, Jessie's current weight is 66 kilograms, and the amount of weight she lost is 126 kilograms. To find her original weight, we can reverse the subtraction by adding the weight

she lost back to her current weight. This is because addition is the inverse operation of subtraction, and it will help us recover the original value.

Let's perform this calculation step by step. Jessie's current weight is 66 kilograms. If we add the 126 kilograms that she lost, we are essentially undoing the weight loss to find out how much she weighed before jogging. Adding these two values together, $66 + 126 = 192$. Therefore, Jessie weighed 192 kilograms before she started jogging.

It's always a good practice to double-check our work. If Jessie originally weighed 192 kilograms and then lost 126 kilograms, subtracting $192 - 126$ indeed equals 66 kilograms, which confirms her current weight. This verification confirms that our solution is correct. Jessie's original weight was 192 kilograms before she began jogging.

The equation is: $66 + 126 = 192$

`\boxed{192}`

C.5 STRATEGYQA

Source and target format. The source record contains a yes/no `question`, a Boolean `answer`, supporting `facts`, and identifying metadata. The teacher receives the question and generates an explanation followed by an explicit Yes/No answer. We normalize the generated answer to a Boolean value and compare it with the original label. After verification, the final-answer line is removed and the explanation replaces `facts`; all other fields are preserved. The builder appends "Therefore, the final answer is" followed by the literal marker `\boxed{Yes}` or `\boxed{No}`, according to the original reference label.

StrategyQA: teacher instruction

```
Determine whether the answer to the given question is Yes or No, and
    provide a concise, fact-based explanation that supports the
    answer.

OUTPUT FORMAT:
- Provide the reasoning explanation.
- End with a separate line containing exactly
  "Final answer: Yes" or "Final answer: No".
- Do not use \boxed{Yes} or \boxed{No}.
```

The following example has the source boolean label `false`. The teacher generates the explanation, and the builder appends the corresponding answer, `No`, to form the supervised completion.

StrategyQA: example question

Hydrogen's atomic number squared exceeds number of Spice Girls?

Reference answer

No (source boolean: `false`)

StrategyQA: native supervised completion

Hydrogen is the first element and has an atomic number of one. To square a number, you multiply it by itself. The Spice Girls has five members.

Therefore, the final answer is `\boxed{No}`

StrategyQA: CoT supervised completion

To determine whether hydrogen's atomic number squared exceeds the number of Spice Girls, we need to first identify hydrogen's atomic number and the number of Spice Girls.

Hydrogen is the first element in the periodic table, so its atomic number is 1. Squaring this value gives $1^2 = 1$.

Next, we consider the number of Spice Girls. The Spice Girls were a famous pop group formed in the 1990s, consisting of five members: Melanie Brown (Scary Spice), Melanie Chisholm (Sporty Spice), Emma Bunton (Baby Spice), Geri Halliwell (Ginger Spice), and Victoria Beckham (Posh Spice). This means there are exactly five Spice Girls.

Now we compare the two values: hydrogen's atomic number squared is 1, and the number of Spice Girls is 5. Clearly, 1 is less than 5. Thus, hydrogen's atomic number squared does not exceed the number of Spice Girls.

Therefore, the final answer is `\boxed{No}`

D Predictive Entropy and Reasoning Difficulty: An Empirical Analysis

Section 3.3 motivates predictive entropy as a model-intrinsic signal for selecting candidate memory-invocation positions. Here we examine its relationship with reasoning structure and difficulty through a training-free analysis of reasoning traces generated by the unaugmented Qwen3-4B-Instruct backbone on 30 GSM8K problems. The analysis considers three aspects: reasoning phases (Section D.2), phase transitions (Section D.3), and empirical problem difficulty (Section D.4).

D.1 Setup

Sampling. We use empirical difficulty labels (each GSM8K training problem is labeled `easy`/`medium`/`hard` by the empirical pass-rate of Qwen3-4B-Instruct over $K{=}8$ samples). We then draw a balanced stratified sample of 30 problems—10 from each difficulty bucket—seeded for reproducibility. No MemGen module is involved; the unaugmented base reasoner generates a single greedy completion per problem with $\max_\text{new}_\text{tokens}{=}1024$, capturing 13,347 output tokens in total.

Per-token signal. For each decoding step t we record the predictive entropy H_t of the next-token distribution (Equation 4 in the main text), the top-1 probability, and the top-k probability gap. All subsequent analyses operate on these per-token records.

Phase classification. We segment each completion into reasoning *lines*—maximal token spans terminated by a newline character—and assign each line one of five phase labels via a deterministic rule-based classifier. The classifier is intentionally lightweight: it uses regular-expression matchers on lexical cues that are characteristic of mathematical chain-of-thought traces, with a digit-density fallback for purely arithmetic lines. The classification rules are summarized below; we provide the verbatim Python implementation in Listing 1 for reproducibility.

- **Conclusion** — lines that introduce or assert a final answer, identified by markers such as `\boxed`, "the (final) answer is", "in total", "altogether", or "therefore ... answer". These markers take precedence over all other rules so that an answer-bearing line is never misclassified as a pure transition.
- **Logic Jump** — lines whose first non-whitespace token is a discourse connective signalling a deductive step (*so, therefore, thus, hence, this/that means, in other words, as a result, consequently, it follows, we can see/conclude/ determine/find, from this*). These are precisely the "transitional" moments at which one substep of reasoning is committed and the next is opened.
- **Setup / Equation** — lines that introduce a variable, a modeling assumption, or a known quantity, identified by leading keywords (*let, suppose, if we, we know, given, define, denote, assume, set*) or by definite-noun-phrase openings (*the total / number / amount / cost / price / rate / ratio / difference*).
- **Calculation** — lines that did not match any of the above and consist primarily of arithmetic. Operationally, a line is labeled Calculation if and only if its digit ratio exceeds 0.25 and it contains either = or at least one of the characters `+`, `−`, `*`, `/`, and `x`.

- **Other** — every line that does not match any of the above. Empirically, this catches narrative connective tissue such as restatements of the problem, summary recaps, and whitespace-only lines.

Phase transitions. A *phase transition* is a boundary between two consecutive lines whose labels differ *and* whose later label is not `Other` (so that drift into narrative filler is not counted as a transition). For each transition we mark its starting token position τ together with the four following tokens $\{\tau{+}1, \ldots, \tau{+}4\}$ as the *transition window*—a five-token region in which the boundary itself, plus the immediate continuation that operationalizes the new phase, are both captured. The complementary set of all remaining tokens forms the *non-transition* set. Across the 30 problems, 172 phase transitions yield 853 transition tokens (6.4% of all tokens) and 12,494 non-transition tokens.

Listing 1: Phase classifier (verbatim Python). Conclusion has highest precedence so that final-answer lines are never misclassified as Transition. Calculation requires *both* a high digit ratio *and* explicit arithmetic syntax, preventing lines that merely reference numerals from being absorbed into the Calculation phase.

```
TRANSITION_RE = re.compile(
    r"^\s*(so|therefore|thus|hence|this means|that means|"
    r"which means|in other words|as a result|consequently|"
    r"it follows|we can (see|conclude|determine|find)|"
    r"from this)\b", re.IGNORECASE)

SETUP_RE      = re.compile(
    r"^\s*(let|suppose|if we|we know|given|define|denote|"
    r"assume|set |the (total|number|amount|cost|price|"
    r"rate|ratio|difference))\b", re.IGNORECASE)

CONCLUSION_RE = re.compile(
    r"(\\boxed|the answer is|the final answer|in total|"
    r"altogether|the result is|we get a final|"
    r"therefore.{0,20}answer)", re.IGNORECASE)

DIGIT_RE = re.compile(r"[0-9]")

def classify_line(line: str) -> str:
    text = line.strip()
    if not text:                          return "other"
    if CONCLUSION_RE.search(text):        return "conclusion"
    if TRANSITION_RE.match(text):         return "transition"
    if SETUP_RE.match(text):              return "setup"
    digit_ratio = len(DIGIT_RE.findall(text)) / max(len(text), 1)
    has_eq = "=" in text
    has_op = any(op in text for op in ["+","-","*","/","x"])
    if digit_ratio > 0.25 and (has_eq or has_op):
        return "calculation"
    return "other"
```

D.2 Finding 1 — Entropy is concentrated in deductive phases, not in arithmetic ones

Figure 2(a) plots the token-entropy distribution within each of the five reasoning phases. The pattern is unambiguous: the two phases that carry the actual reasoning load—Logic Jump (μ=0.379, n=938) and Setup / Equation (μ=0.333, n=410)—exhibit substantially higher entropy than the phase that merely *executes* a committed plan, Calculation (μ=0.025, n=1,086). The Logic-Jump-to-Calculation ratio is $15\times$. Conclusion also exhibits relatively low entropy ($\mu = 0.092$), consistent with lower predictive uncertainty when stating an already-derived answer. The Other phase sits in the middle (μ=0.196).

This contrast shows that predictive uncertainty varies substantially across reasoning phases. Calculation tokens have low average entropy, whereas Logic Jump and Setup / Equation tokens exhibit much higher values. Entropy-based gating can therefore distinguish candidate positions by the Reasoner's predictive state, including non-delimiter positions within a clause that punctuation-based gating would not consider.

D.3 Finding 2 — Entropy spikes at phase transitions

Finding 1 establishes a difference between phases as wholes; we next ask whether the same signal localizes the *boundaries* between phases. Figure 2(b) compares the entropy distribution of the 853 transition-window tokens against that of the 12,494 non-transition tokens. Two observations stand out. First, the non-transition distribution is sharply concentrated near zero, whereas the transition distribution exhibits a substantially heavier upper tail between 0.5 and 1.5 nats. Second, the means differ by $+67.5\%$ ($\mu_{\text{transition}}$=0.315 vs. $\mu_{\text{non-transition}}$=0.188); a Mann-Whitney U test of the one-sided alternative $H_{\text{transition}} > H_{\text{non-transition}}$ returns $p < 10^{-3}$, confirming that the difference is not an artifact of distributional skew. Note that this is a strictly more demanding test than Finding 1: rather than comparing aggregate phase statistics, it asks whether H_t separates a sparse 6.4% minority of tokens (the transition windows) from the dominant majority. The result is that it does, by a wide statistical margin.

Together, Findings 1 and 2 show that H_t is sensitive to the same two structural features of a reasoning trace at two complementary granularities: *within* phases (Logic Jump $\gg$ Calculation) and *between* phases (transition tokens $\gg$ non-transition tokens).

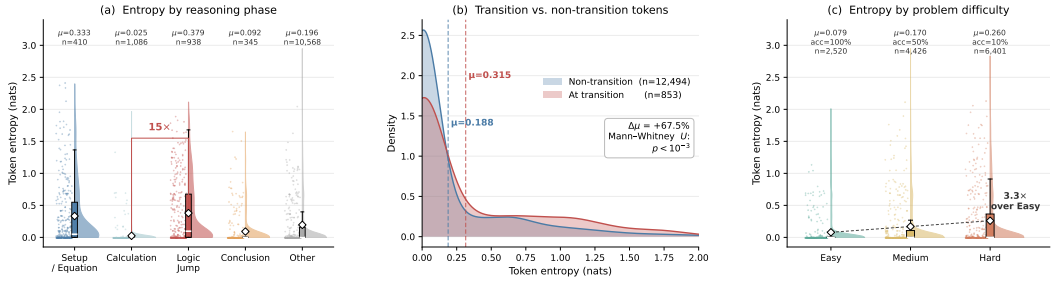


Figure 2: **Predictive entropy is a multi-scale reasoning-difficulty signal** (30 GSM8K problems, Qwen3-4B-Instruct, no MemGen). **(a)** Token-entropy distribution stratified by reasoning phase. Logic Jump tokens carry $15\times$ the mean entropy of Calculation tokens, mirroring the intuition that committing to a new substep is harder than executing an already committed one. Distributions are shown as raincloud plots (jittered scatter + half-violin + box; mean marked $\diamond$). **(b)** Entropy at phase-transition tokens (n=853, the 5-token windows that follow each phase boundary) versus all other tokens (n=12,494). The transition distribution has a $+67.5\%$ higher mean and a substantially heavier upper tail; one-sided Mann–Whitney U rejects equality at $p < 10^{-3}$. **(c)** Per-token entropy by source-problem difficulty (10 problems per bucket). Entropy increases monotonically with empirical hardness (`easy` < `medium` < `hard`, $3.3\times$ overall); base-model accuracy is annotated above each violin. **Take-away.** Predictive entropy varies with reasoning phase, phase transitions, and problem difficulty, providing a model-intrinsic basis for selecting candidate memory-invocation positions.

D.4 Finding 3 — Entropy tracks problem-level difficulty

A signal that localizes within-trace difficulty would still be of limited value if it failed to scale with the global hardness of the problem. Figure 2(c) stratifies the per-token entropy distribution by the empirical difficulty bucket of the source problem. The mean entropy increases monotonically from `easy` (μ=0.079, base accuracy 100%) through `medium` (μ=0.170, accuracy 50%) to `hard` (μ=0.260, accuracy 10%); the `hard`-to-`easy` ratio is $3.3\times$. Crucially, this ordering also recovers *model correctness*: among the 30 completions, the 16 correct ones have a mean per-question entropy of 0.085, while the 14 incorrect ones have 0.214—a $2.5\times$ ratio (Welch's t=3.73, p=0.002;

Figure 4). In other words, the same entropy signal that flags within-trace transitions also predicts, at the trace level, whether the reasoner will succeed.

D.5 Implication for RSAT

The three findings show that predictive entropy reflects variation across reasoning phases, phase transitions, and problem difficulty. Together, they motivate its use as a model-intrinsic gating signal for RSAT. Whereas punctuation-based gating selects candidates using delimiter tokens, RSAT selects them using the Reasoner's current predictive uncertainty. Section 4.4 evaluates the downstream performance of this choice.

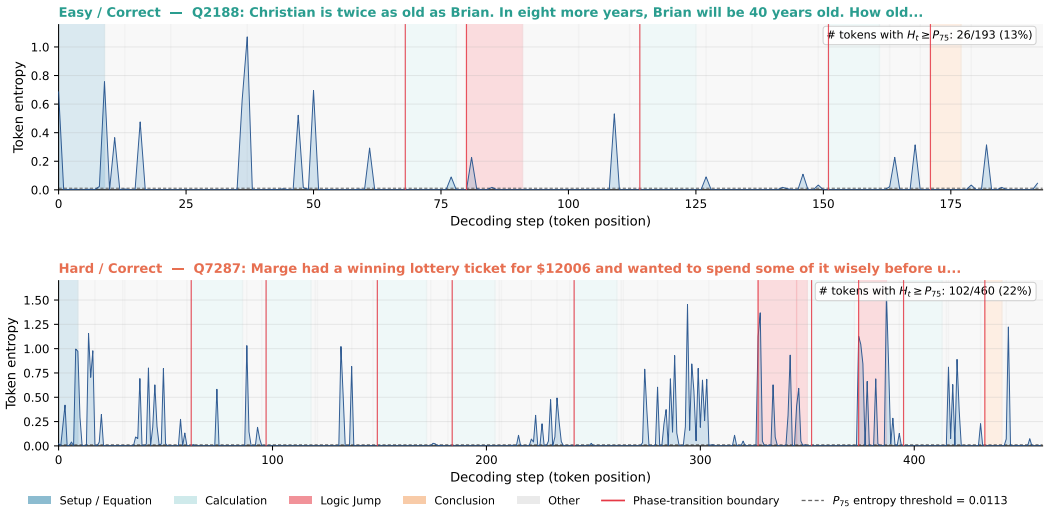


Figure 3: **Token-level entropy traces for two representative problems.** Background colors indicate reasoning phase; vertical red lines mark phase-transition boundaries; the dashed horizontal line is the P_{75} entropy threshold computed on the `gsm8k_enhanced` training set (τ=0.011). **Top:** An easy/correct problem produces few above-threshold tokens (13%) and these are tightly clustered around phase boundaries. **Bottom:** A hard/correct problem produces considerably more above-threshold tokens (22%), distributed in bursts that align with the multiple Logic-Jump phases. The contrast illustrates how a single, fixed entropy threshold automatically allocates more recall opportunities to harder problems.

E Qualitative Comparison of Compressed and Uncompressed Memory

We compare the reasoning trajectories produced with uncompressed CoT memory and CSMC on a SVAMP example. Both configurations use the same CoT-supervised Weaver. The comparison focuses on how the Reasoner interprets the quantities in the problem and forms its final answer.

Problem. John uses 12 large wooden planks and 10 small wooden planks to build a wall. The 12 large planks together require 15 nails, and the 10 small planks together require 5 nails. The question asks for the total number of nails required. The correct calculation is

$$15 + 5 = 20. \tag{7}$$

Uncompressed CoT memory. With uncompressed CoT-supervised memory, the Reasoner incorrectly interprets 15 and 5 as the numbers of nails required per plank. It therefore produces the following reasoning:

> The 12 large planks require $12 \times 15 = 180$ nails, and the 10 small planks require $10 \times 5 = 50$ nails. Therefore, the total number of nails is $180 + 50 = 230$.

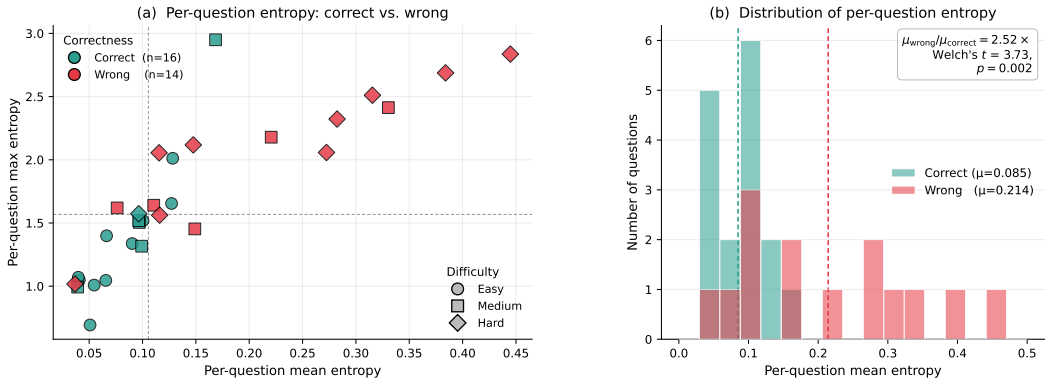


Figure 4: **Per-question entropy summary: correct vs. incorrect completions.** **(a)** Each marker is one of the 30 problems; color encodes correctness, shape encodes empirical difficulty. Dashed lines mark medians along each axis. The upper-right quadrant (high mean and max entropy) is dominated by `hard`/wrong cases; the lower-left quadrant by `easy`/correct cases. **(b)** Histogram of per-question mean entropy for the two correctness groups. Wrong-answer problems have $2.5\times$ the mean entropy of correct-answer problems (Welch's t=3.73, p=0.002), confirming that entropy is a trace-level success predictor and not only a within-trace localizer.

This trajectory is internally consistent after the initial misinterpretation, but it ignores the scope of "together" in the problem statement and consequently predicts 230.

CSMC. With compressed memory, the Reasoner instead preserves the aggregate meaning of the two quantities:

> The 12 large planks together require 15 nails, and the 10 small planks together require 5 nails. Therefore, the total number of nails is $15 + 5 = 20$.

The resulting prediction is correct.

Interpretation. The uncompressed configuration develops a coherent calculation from an incorrect per-plank interpretation, whereas CSMC produces a reasoning trajectory that correctly treats the nail counts as group totals. The contrast illustrates a case in which compressed memory supports a more accurate interpretation under the same CoT supervision. This is consistent with the goal of task-driven compression: providing a compact memory that supports downstream reasoning.