Unlocking the Working Memory of Large Language Models for Latent Reasoning

Lukas Aichberger [1] **Sepp Hochreiter** [1,2]

[1] ELLIS Unit Linz and LIT AI Lab, Institute for Machine Learning,
Johannes Kepler University Linz, Austria
[2] NXAI GmbH, Linz, Austria
{aichberger, hochreit}@ml.jku.at

Abstract

To improve the reasoning capabilities of large language models, test-time compute is typically scaled by generating intermediate tokens before the final answer. However, this couples reasoning to autoregressive generation and thereby conflates internal computation with external communication. In contrast, human cognition can use working memory to hold and manipulate information internally without the need to externalize intermediate thoughts. Drawing on this principle, we introduce *Reasoning in Memory (RiM)*, a latent reasoning method that replaces the autoregressive generation of reasoning steps with memory blocks. These memory blocks are fixed sequences of special tokens that unlock the working-memory capacity of large language models. Since they are fixed rather than generated, they can be processed in a single forward pass, enabling compute-efficient latent reasoning. To operationalize these memory blocks, we employ a two-stage curriculum. First, we ground them by predicting explicit reasoning steps after each memory block. Second, we discard this step-level supervision and iteratively refine the final answer after each memory block. Our experiments on reasoning benchmarks show that, across language models of different families and sizes, RiM matches or exceeds existing latent reasoning methods while avoiding the autoregressive generation of thoughts. These results demonstrate that large language models can be trained to use working memory as an effective mechanism for latent reasoning.

1 Introduction

Large language models (LLMs) have demonstrated remarkable reasoning capabilities, primarily driven by techniques that scale test-time compute through generating intermediate tokens before giving the final answer [Wei et al., 2022, Snell et al., 2024]. In chain-of-thought (CoT) reasoning, intermediate computation remains directly coupled to the generation of text. This coupling forces an LLM to "think out loud", anchoring its reasoning process to the syntax and structure of natural language. However, language is optimized for communication rather than for computation [Wei et al., 2022, Kojima et al., 2022]. Consequently, part of the computational budget is allocated for generating grammatical and fluent intermediate text, rather than being devoted purely to internal computation.

Recent advances in latent reasoning bypass these natural-language constraints by replacing discrete tokens with continuous representations [Hao et al., 2025, Shen et al., 2025, Cheng and Durme, 2024]. However, they preserve the same step-by-step generation paradigm, effectively "thinking out loud" in a continuous space. Whether generating discrete tokens or continuous representations, each intermediate computation must still be externalized before future computation can condition on it. Thus, these latent reasoning methods preserve the coupling of intermediate computation to autoregressive generation.

Preprint.

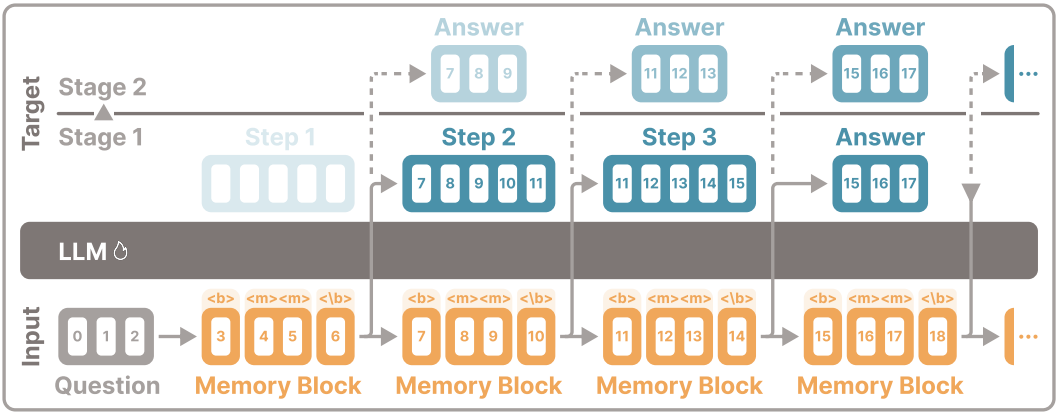


Figure 1: **Reasoning in Memory (RiM).** Stage 1 trains the LLM to use memory blocks (yellow) as working memory by supervising the prediction of the next reasoning step (blue) after each memory block. Once the memory blocks are grounded for intermediate computation, Stage 2 removes reasoning-step supervision and trains the LLM to refine the final answer after each memory block.

Human cognition suggests a different design principle. When solving complex problems, humans do not typically articulate every intermediate step, nor is their reasoning strictly bound to the syntax of language. Instead, cognitive psychology identifies *working memory* as an internal workspace for holding and manipulating task-relevant representations [Baddeley, 1992]. Developmental psychology observes that while we initially rely on "thinking out loud", this linguistic scaffolding is eventually abandoned in favor of internal processing [Vygotsky, 1978]. This motivates a shift from generating intermediate reasoning steps autoregressively to representing them in a dedicated internal workspace of an LLM, analogous to human working memory.

Drawing on these cognitive principles, we introduce *Reasoning in Memory (RiM)*, which operationalizes *memory blocks*, fixed sequences of special tokens, as a working memory for latent reasoning. Rather than generating intermediate reasoning steps autoregressively, our method trains the LLM to use these memory blocks as an internal latent workspace for task-relevant computation. While their token identities and positions are fixed, their contextual representations remain task-dependent and can encode intermediate reasoning. A direct benefit of these fixed memory blocks is that they can be processed in a single forward pass. This avoids the bottleneck of autoregressively generating discrete tokens or continuous representations, making our method highly parallelizable and compute-efficient.

In order to teach the LLM to use these memory blocks as a latent workspace, RiM employs a two-stage curriculum (Figure 1). Since the memory blocks have no predefined computational role, unlocking their working-memory capacity requires a carefully structured training signal. In Stage 1, we therefore ground the memory blocks by supervising the readout after each block to predict the corresponding next reasoning step. As each readout can recover this target only from memory blocks seen so far, the LLM is forced to structure the latent workspace around task-relevant intermediate information. In Stage 2, once the working-memory capacity is learned, we remove this strict step-level supervision and instead supervise the readout after each block to predict the final answer. The training signal is therefore shifted from predicting written reasoning steps to progressively refining the answer across memory blocks. A custom attention mask (Figure 2) allows all readouts to be trained in a single forward pass, making supervision dense and efficient in both stages (see Section D for details).

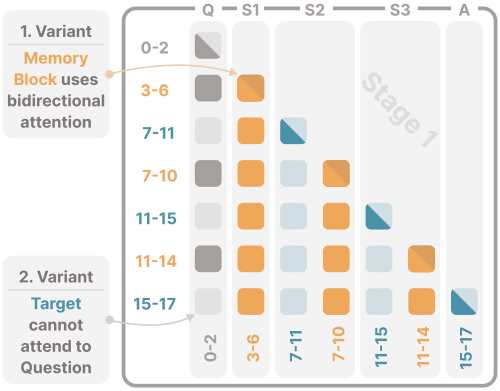


Figure 2: **RiM Attention Mask.** Memory blocks (yellow) attend to the question and previous memory blocks. Target reasoning steps (blue) attend to previous memory blocks and optionally the question, but not to other reasoning steps. This enables all targets to be predicted in one forward pass without information leakage, forcing reasoning inside the memory blocks.

Empirically, we validate RiM by training language models across GPT-2 and Llama-3.2 scales on GSM8K-Aug and evaluating them on GSM8K and GSM-Hard as in- and out-of-distribution benchmarks, respectively. Across these settings, RiM matches or exceeds existing CoT and latent reasoning baselines. These findings show that our method enables LLMs to use working memory for efficient reasoning by combining dense supervision during training with compute-efficient inference.

Our main contributions are as follows:

- We introduce Reasoning in Memory (RiM), a latent reasoning method that unlocks the working-memory capacity of LLMs by providing a latent workspace for intermediate computation.
- We propose a two-stage curriculum that assigns memory blocks a computational role through carefully structured training signals, enabling their use as working memory for latent reasoning.
- We show that RiM matches or exceeds explicit and latent reasoning baselines across model scales and reasoning benchmarks, while making intermediate computation parallelizable and efficient.

2 Related Work

CoT reasoning. The modern test-time-compute paradigm grew out of methods that place a textual workspace between the question and the answer [Nye et al., 2021]. Chain-of-thought (CoT) prompting [Wei et al., 2022, Kojima et al., 2022] showed that language models solve harder tasks when intermediate computation is externalized as text. First, explicit reasoning has been scaled at inference time by repeated sampling [Brown et al., 2024], aggregating multiple reasoning traces [Wang et al., 2023], structured search [Yao et al., 2023], or process reward models that verify intermediate steps [Lightman et al., 2024, Snell et al., 2024]. Second, it has also been scaled at training time through supervised fine-tuning on self-generated [Zelikman et al., 2022] or ground-truth [Muennighoff et al., 2025] reasoning traces, or through reinforcement learning [DeepSeek-AI, 2025]. These methods share that intermediate computation is externalized by autoregressively generating text.

Explicit latent reasoning. To avoid the syntactic overhead of explicit reasoning, recent work explores latent reasoning. One line of work performs vertical latent reasoning by iteratively refining activations through recurrent modules [Wang et al., 2025a, Jolicoeur-Martineau, 2025]. A second line of work performs horizontal latent reasoning by replacing parts of the reasoning trace with continuous representations along the sequence dimension. Coconut [Hao et al., 2025] is the closest latent analogue of CoT, as it replaces discrete reasoning tokens with continuous representations that are fed back autoregressively. Related methods modify how these continuous representations are trained or used, for example through compressed contemplation tokens [Cheng and Durme, 2024], distillation from explicit CoT [Shen et al., 2025], synthetic continuous targets [Wang et al., 2025b], sparse-autoencoder concepts [Tack et al., 2025], or a hidden Markov model over continuous reasoning variables [Liu et al., 2025]. These methods share that intermediate computation is externalized by autoregressively generating continuous representations.

Other methods are at the intersection of explicit and implicit reasoning. For instance, Kim et al. [2025] dynamically insert pause tokens at low-confidence positions, Sun et al. [2025] attach filler tokens to generated tokens, and Zhang et al. [2025] dynamically compress generated reasoning steps into continuous space. While these methods reduce or redistribute the cost of explicit reasoning, they still operate within the standard autoregressive generation paradigm.

Implicit latent reasoning. Another line of work studies whether intermediate computation can be allocated to tokens without predefined semantic content, which we collectively refer to as filler tokens. Prior work shows that making filler tokens useful for reasoning is difficult. Lanham et al. [2023] find that simply adding filler tokens does not improve accuracy and can even reduce it in longer-context settings. Pfau et al. [2024] then show that filler tokens can support computation on synthetic algorithmic tasks, but require specific dense supervision to converge. Goyal et al. [2024] scale this to real-world downstream tasks, but find that gains arise mainly when filler tokens are used during both pretraining and finetuning. More recently, DART shows that filler tokens can be trained without specific pretraining, but finetuning requires a two-pathway self-distillation framework with an auxiliary "Reasoning Evolvement Module" and multiple auxiliary losses [Deng et al., 2024]. Together, these results suggest that filler-token reasoning is possible, but requires a carefully designed training signal. Our method improves upon this line of work by showing that filler tokens inside fixed memory blocks can be trained as an effective latent workspace for reasoning, which we describe next.

3 Reasoning in Memory

In this section, we introduce *Reasoning in Memory (RiM)*, our method that uses memory blocks as a latent workspace for implicit latent reasoning. We first formalize the sequential generation paradigm underlying chain-of-thought (CoT) and explicit latent reasoning methods, and then present how we can train models to effectively decouple intermediate computation from autoregressive generation.

CoT reasoning. Formally, let $\boldsymbol{x} = (x_1, \ldots, x_q)$ and $\boldsymbol{y} = (y_1, \ldots, y_a)$ denote the discrete token sequences of the input question and the final answer, respectively, and let $\boldsymbol{w}$ denote the parameters of an autoregressive language model. Standard CoT reasoning scales test-time computation by generating a sequence of intermediate reasoning steps $\boldsymbol{r}_{1:T} = (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_T)$ before the final answer [Wei et al., 2022]. Each reasoning step $\boldsymbol{r}_t$ is itself a discrete token sequence generated autoregressively. If the full reasoning trace is flattened into a token sequence $\boldsymbol{r} = (r_1, \ldots, r_C)$, then each reasoning token is decoded from the language model's predictive distribution $p_{\boldsymbol{w}}(r_i \mid \boldsymbol{x}, \boldsymbol{r}_{<i})$. Thus, generating the reasoning trace requires C sequential decoding steps.

Explicit latent reasoning. Latent reasoning methods such as Coconut [Hao et al., 2025] replace written reasoning steps with continuous representations. Instead of decoding intermediate reasoning steps $\boldsymbol{r}_{1:T}$, the model generates a sequence of continuous representations $\boldsymbol{z}_{1:L} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_L)$, with $\boldsymbol{z}_\ell \in \mathbb{R}^d$. Unlike discrete tokens, these continuous representations are not sampled from a predictive distribution. Rather, each $\boldsymbol{z}_\ell$ is obtained from the hidden state at a chosen layer of the language model, conditioned on the question $\boldsymbol{x}$ and the previously generated continuous representations $\boldsymbol{z}_{<\ell}$. The resulting representation is then fed back as the input embedding at the next decoding step. Thus, explicit latent reasoning with continuous representations requires L sequential decoding steps, typically with $L < C$. However, since each representation $\boldsymbol{z}_\ell$ must be generated before $\boldsymbol{z}_{\ell+1}$ can condition on it, intermediate computation remains coupled to autoregressive generation.

Our method. RiM builds on the central advantage of explicit latent reasoning, namely that intermediate computation need not be expressed in natural language, but can instead take place in a continuous space. However, instead of explicitly generating continuous representations and feeding them back into the language model, RiM moves this computation into fixed working memory. Concretely, we provide the language model with K fixed memory blocks $\boldsymbol{m}_{1:K} = (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_K)$, where each memory block consists of a predefined sequence of special tokens

$$\boldsymbol{m}_k = [\texttt{}, \underbrace{\texttt{<m>}, \ldots, \texttt{<m>}}_{M \text{ tokens}}, \texttt{}] \,. \tag{1}$$

The idea is that <m> tokens form the latent workspace, while and delimit block boundaries. Although memory blocks could in principle be constructed from existing vocabulary tokens, we introduce dedicated special tokens to avoid overloading pretrained token semantics. Additionally, to minimize interference with the pretrained vocabulary, we freeze the embeddings of all existing tokens and update only the embeddings of the special tokens. This isolates adaptation to the newly introduced special tokens, ensuring that improvements arise from learning to use the memory blocks rather than from altering pretrained token representations. The special tokens are otherwise treated as standard input tokens and are processed by the language model in the usual way. Unless stated otherwise, we fix $M = 2$ in all experiments. The fixed memory blocks $\boldsymbol{m}_{1:K}$ are appended to the input question $\boldsymbol{x}$ and serve as a latent workspace for intermediate computation. Since the memory blocks are fixed input tokens, the language model processes the entire augmented sequence $(\boldsymbol{x}, \boldsymbol{m}_{1:K})$ in a single forward pass. Thus, RiM retains the central benefit of reasoning in continuous space while avoiding the sequential bottleneck of generating continuous representations one at a time.

Learning to reason in memory. As memory blocks have no predefined computational role, eliciting useful latent computation is a central challenge. Without targeted supervision, the language model may simply ignore them or treat them as distracting context, rather than using them to hold and manipulate task-relevant information [Lanham et al., 2023, Pfau et al., 2024, Goyal et al., 2024]. Thus, effective reasoning in memory requires a carefully structured supervision signal that teaches the model how to use these blocks as working memory.

To combine training efficiency with dense supervision, RiM uses a simple two-stage curriculum. Both stages use the same simple objective, standard next-token prediction, but with different targets. This avoids multiple pathways and auxiliary objectives used by distillation-based latent reasoning methods [Jiang et al., 2025, Shen et al., 2025], keeping optimization simple while still providing dense supervision over the latent workspace. We now describe the objectives for the two stages.

3.1 Stage 1: Reasoning Step Supervision

The goal of Stage 1 is to teach the model to use memory blocks as a latent workspace. We achieve this by grounding them with explicit reasoning-step supervision. After each memory block, the model is trained to recover the next written reasoning step from the latent workspace available to it. This encourages task-relevant information to be stored and transformed through the memory blocks. Stage 1 is closely aligned with JEPA-style predictive representation learning because fixed memory blocks learn to predict missing reasoning structure, turning them into parallel latent states for working memory [LeCun, 2022, Assran et al., 2023].

A naive adaptation of Coconut-style staging [Deng et al., 2024, Hao et al., 2025] would gradually replace written reasoning steps with memory blocks. In preliminary experiments, however, we found that this alone does not provide a sufficiently dense training signal. First, during the early stages, target reasoning steps can attend to previous reasoning steps, allowing the model to predict the target directly from the preceding reasoning steps. The supervision signal can therefore bypass the latent workspace, leaving the memory blocks with a weak training signal. Second, Coconut-style staging requires fixing in advance a maximum number of training stages. It can therefore replace only that many reasoning steps with latent representations before switching to final-answer prediction. Consequently, some reasoning steps would never be directly supervised from the memory blocks.

RiM avoids these shortcomings by replacing the written reasoning trace one-to-one with memory blocks during Stage 1. For a trace with T reasoning steps, we allocate T memory blocks and supervise each block-level readout against the next reasoning step (Figure 3). To enable this dense supervision in a single forward pass, we employ a custom attention mask so each readout can attend to the memory blocks, and optionally to the question, but not to other written reasoning steps (Figure 2). As a result, every reasoning step provides a direct supervision signal through the latent workspace. The model cannot recover it from previous written steps and must instead encode the necessary intermediate computation in the memory blocks themselves. We refer to Section D for details.

Formally, for a training sample $(\boldsymbol{x}, \boldsymbol{r}, \boldsymbol{y})$ consisting of token sequences corresponding to the question, reasoning trace, and final answer, respectively, we first segment the reasoning trace into T reasoning steps $\boldsymbol{r}_{1:T} = (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_T)$. We then append one memory block per reasoning step and train the readout after block $\boldsymbol{m}_t$ to predict the next reasoning step $\boldsymbol{r}_{t+1}$, where we define $\boldsymbol{r}_{T+1} = \boldsymbol{y}$ for notational convenience. The Stage 1 objective is a weighted reasoning-step negative log-likelihood,

$$\mathcal{L}_{\mathrm{S1}}(\boldsymbol{w}) = -\sum_{t=1}^{T} \lambda_t(s) \log p_{\boldsymbol{w}}(\boldsymbol{r}_{t+1} \mid \boldsymbol{x}, \boldsymbol{m}_{\leq t}) , \tag{2}$$

where $\lambda_t(s) \in [0, 1]$ controls the supervision strength for the readout after memory block $\boldsymbol{m}_t$, at training step s. Annealing $\lambda_t(s)$ yields a soft multi-stage reasoning curriculum in which all readouts receive dense supervision early in training, before supervision is gradually removed from earlier reasoning steps first and later reasoning steps last. We experimented with several weighting schedules and found that relative weighting with respect to the number of reasoning steps T in each sample performs best. Absolute weighting with respect to the maximum number of reasoning steps removes supervision too early for shorter samples, while no weighting remains slightly worse because dense supervision is never relaxed. We therefore use a linear relative schedule throughout our experiments, where $\lambda_t(s)$ decreases linearly from 1 to 0 over Stage 1, with the annealing order determined by t.

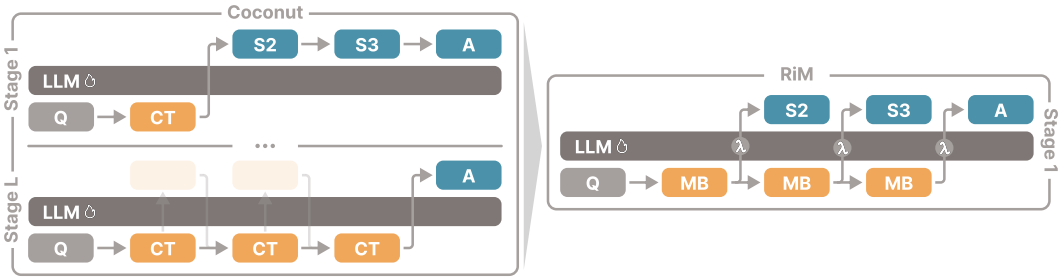


Figure 3: **RiM Stage 1.** While Coconut [Hao et al., 2025] uses multiple curriculum stages to train continuous thoughts (CTs), progressively increasing the number of steps, RiM collapses this into a single stage over all memory blocks (MBs), forcing dense supervision through the latent workspace.

3.2 Stage 2: Final Answer Refinement

After Stage 1, the language model has been trained to route intermediate computation through memory blocks using written reasoning steps as supervision. Stage 2 removes this intermediate supervision. Instead of supervising each memory block to recover a reasoning step, the model is trained to directly predict and refine the final answer after each memory block. We use the same custom causal mask as in Stage 1, so each readout can attend only to the question and the memory blocks available up to that point. While Stage 1 uses one memory block per reasoning step to provide dense supervision, Stage 2 uses a fixed number of K memory blocks and trains the model with an anytime answer objective, matching the inference-time setting more closely. Each readout predicts the final answer from the memory blocks available up to that point, encouraging the model to use the learned latent workspace to improve its prediction as the available memory budget increases.

Formally, given the same training sample $(\boldsymbol{x}, \boldsymbol{r}, \boldsymbol{y})$, we discard the written reasoning trace $\boldsymbol{r}$ and use a fixed number of K memory blocks for every sample. After each memory block $\boldsymbol{m}_k$, we attach an answer readout and train it to predict the final answer $\boldsymbol{y}$. The Stage 2 objective is a weighted final-answer negative log-likelihood,

$$\mathcal{L}_{\mathrm{S2}}(\boldsymbol{w}) \;=\; -\sum_{k=1}^{K} \alpha_k \, \log p_{\boldsymbol{w}}(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{m}_{\leq k}) \,, \tag{3}$$

where $\alpha_k \in [0, 1]$ controls the supervision strength for the readout after memory block $\boldsymbol{m}_k$, independent of the training step. While this stage is relatively insensitive to the exact weighting schedule, we found that assigning larger weights to later memory blocks performs slightly better. We again use a linear weighting schedule throughout our experiments, reflecting that later readouts have access to more latent computation and therefore should produce the strongest final answers.

Stage 2 is inspired by iterative latent reasoning models such as HRM [Wang et al., 2025a] and TRM [Jolicoeur-Martineau, 2025], but realizes refinement horizontally along the sequence dimension. Additional memory blocks provide additional latent computation before the answer readout, without requiring recurrent refinement modules or autoregressively generated latent states. In this sense, Stage 2 converts the grounded memory blocks from Stage 1 into a fixed sequence of latent computations for final-answer prediction.

Following Deng et al. [2024], Hao et al. [2025], we reset the optimizer state and learning-rate scheduler at the stage switch. We also use a lower learning rate and increased dropout to reduce overfitting under dense answer supervision. This hard switch is important because the two stages optimize distinct objectives: Stage 1 grounds memory blocks for intermediate computation, whereas Stage 2 uses this learned latent workspace to refine the final answer under a fixed memory budget.

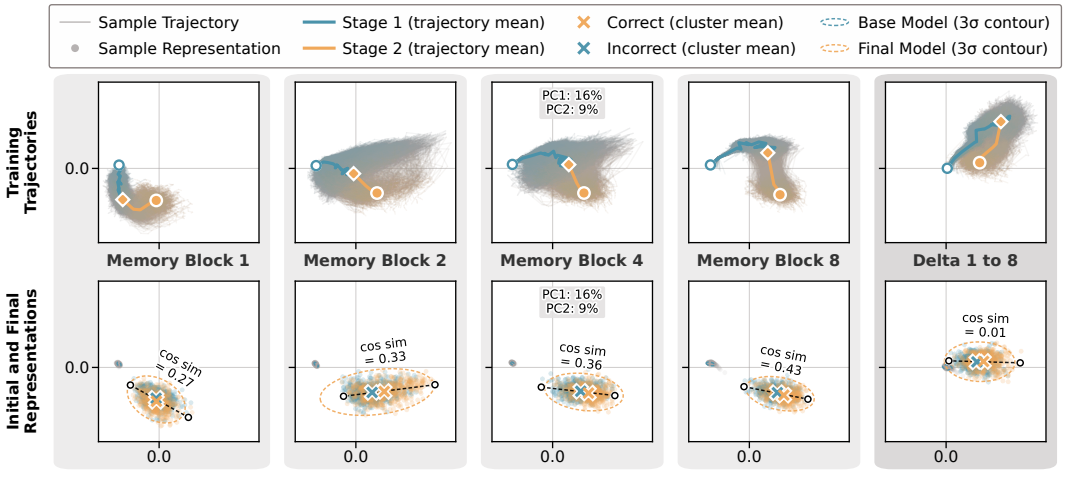


Figure 4: **Memory block representations.** Using all GSM8K test questions, we project memory block representations and the first-to-final memory block representation delta into a shared PCA basis. The top row shows their trajectories during training. The bottom row shows the representations from the initial base model (Llama-3.2-1B) and the final RiM-trained model in the same PCA basis.

4 Experiments

Having introduced Reasoning in Memory (RiM), we now evaluate whether fixed memory blocks provide an effective mechanism for latent intermediate computation. We focus on three questions: First, does RiM induce the language model to use memory blocks for intermediate computation? Second, how does RiM compare to prior reasoning methods in terms of performance and latency? Third, is the performance of RiM robust across different inference-time memory budgets?

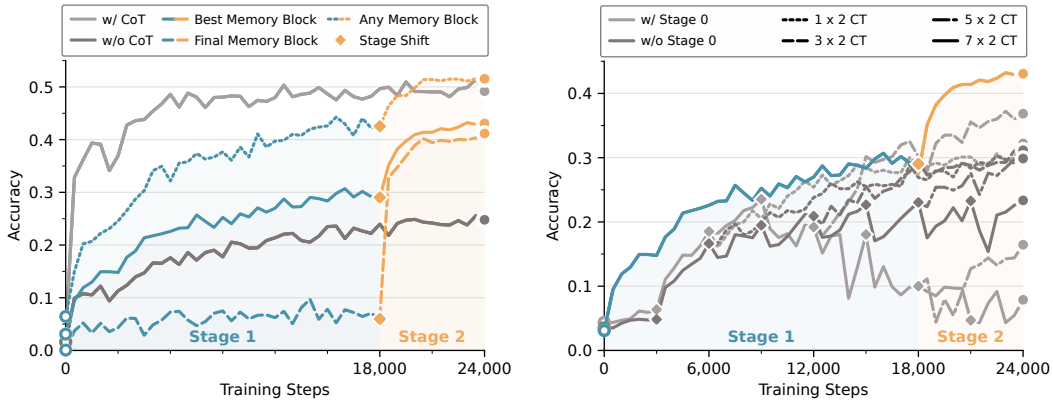
Models and Datasets. To answer these questions, we use the models and datasets adopted in established prior work on latent reasoning [Deng et al., 2024, Hao et al., 2025, Shen et al., 2025, Goyal et al., 2024, Jiang et al., 2025], enabling direct comparison with established methods. For training, we use *GSM8K-Aug* [Deng et al., 2023], a dataset of 386K grade-school math questions with up to 13 reasoning steps provided as mathematical expressions. For evaluation, we use *GSM8K* [Cobbe et al., 2021] as the in-distribution (ID) test set and *GSM-Hard* [Gao et al., 2023] as an out-of-distribution (OOD) test set, consisting of more challenging math questions. We evaluate across *GPT-2* [Radford et al., 2018] and *Llama-3.2* [Dubey et al., 2024] model scales, covering the two most common model families for latent reasoning. Together, these evaluations cover transfer across model families, scaling across parameter sizes, and generalization from ID to OOD reasoning problems.

Baselines. We compare RiM against the most directly related reasoning baselines. First, we include supervised fine-tuning (SFT) [Muennighoff et al., 2025] in two variants. *SFT w/o CoT* is trained on question-answer pairs to directly predict the answer, making it the closest non-latent baseline to RiM. *SFT w/ CoT* is trained on explicit reasoning traces to generate a full CoT before predicting the answer, making it a strong explicit-reasoning baseline that incurs substantially higher inference cost. Second, we include Coconut [Hao et al., 2025], the most widely used latent reasoning baseline, which replaces CoT reasoning with autoregressively generated CTs. *Coconut w/ Stage 0* follows the original training recipe and gives Coconut a supervised warm start on explicit reasoning traces, while *Coconut w/o Stage 0* omits this initial stage. Third, we compare against DART [Jiang et al., 2025] using the official reported numbers, since no official codebase is available and our reimplementation did not yield a reliable reproduction. This comparison favors DART in terms of training compute, since DART uses more epochs and a two-pathway training objective. Nevertheless, RiM outperforms the reported DART results in all comparable settings, as discussed in Section D.

Evaluation. For SFT w/o CoT as well as all latent reasoning methods, we force the answer prefix "`The final answer is \boxed{`", ensuring they are evaluated only on final-answer generation. In evaluating performance, prior work usually evaluates models at multiple training checkpoints and reports the checkpoint with the highest accuracy on the evaluation benchmark itself. This introduces selection overfitting and can overstate performance [Cawley and Talbot, 2010]. Thus, to separate selection from evaluation, we use k-fold cross-validation with a predefined checkpoint-selection protocol. For each of the 16 splits, we reserve 264 held-out GSM8K samples for checkpoint selection, choosing the checkpoint with the highest greedy accuracy for each model-method combination. Unless stated otherwise, these selected checkpoints are used throughout the following analyses to answer the three questions.

4.1 RiM Induces Latent Computation in Memory Blocks

We first examine whether the language model learns nontrivial contextual representations at the memory-block positions. If the model ignored the memory blocks, or used them only as fixed placeholders, their contextual representations should remain largely determined by token identity and position, with little systematic dependence on the input or on training. By contrast, if the memory blocks served as a useful latent workspace, their contextual representations should change systematically during training and become input-dependent. To assess which case applies, we train Llama-3.2-1B for 6 epochs in Stage 1 with up to 13 memory blocks and 2 epochs in Stage 2 with 8 memory blocks, saving checkpoints every 1,000 training steps. This yields 18 and 6 checkpoints, respectively, at which we collect the penultimate-layer representation of each memory block per GSM8K test question. We then project all representations into a shared PCA basis, which explains 25% of the total variance. Figure 4 shows the representations of individual memory blocks as well as the delta between the first and the final memory block. For each question, this delta is computed in the original representation space before being projected into the same PCA basis, controlling for question-specific offsets in the representation space and thus isolating how the representations evolve across memory blocks rather than reflecting differences between questions.



(a) RiM vs. SFT.

(b) RiM vs. Coconut.

Figure 5: **Llama-3.2-1B training curves.** Greedy accuracy on GSM8K test questions over training, comparing RiM to SFT and Coconut. RiM is trained for 6 epochs in Stage 1 and 2 epochs in Stage 2, while Coconut is trained with 1 to 7 stages and 2 continuous thoughts (`x 2 CT`) added per stage.

Representation trajectories during training. In the top row of Figure 4, we plot how the representations change during training, with a separate trajectory for each question as well as the mean trajectory of all questions. We observe that these trajectories are block-specific and diverge as training progresses, which indicates that the model learns to make the latent workspace input dependent. Also, the mean trajectories are smooth, which indicates that training systematically organizes the representations rather than merely perturbing them.

Representations from base and final model. In the bottom row of Figure 4, we plot the memory-block representations of the base model before training and the final model after training. These representations correspond to the start and end points of the trajectories shown in the top row. While the base-model representations are largely collapsed, the final-model representations spread out and exhibit question-specific structure, suggesting that different questions induce distinct latent workspaces. For the two samples that are farthest apart in the PCA projection, we additionally compute their cosine similarity in the original representation space. The low similarity confirms that the separation observed in the PCA projection reflects a corresponding difference in the original representations. This suggests that different questions induce distinct directions in the latent workspace, and that these differences are faithfully reflected by the PCA projection. Together, these results show that memory blocks become both block-specific and sample-dependent, providing representation-level evidence that RiM learns to use them as a latent workspace. Plots for all 8 memory blocks and for representations from different layers of the language model are provided in Figure 11.

4.2 RiM Achieves High Performance at Low Latency

Next, we examine whether the structured memory-block representations translate into strong final-answer accuracy and whether RiM improves the accuracy-latency trade-off at inference time.

Performance during training. We first ask whether the structured memory-block representations observed above translate into improved task performance. For RiM, we distinguish three readouts. Final-block accuracy corresponds to the deployable fixed-budget setting used in the main comparison below, best-block accuracy reports the accuracy of the single best memory block, and any-block accuracy counts a sample as correct if any memory-block readout produces the correct answer. Figure 5 tracks these readouts through training for Llama-3.2-1B on GSM8K. Figure 5a compares RiM to the two SFT variants. The final-block readout improves sharply after the transition to Stage 2 and exceeds SFT w/o CoT, while any-block accuracy becomes competitive with SFT w/ CoT despite not generating explicit reasoning traces. Figure 5b compares RiM to Coconut variants with different latent reasoning budgets. Although additional latent stages give Coconut more sequential latent computation, they do not improve performance. The strongest Coconut variant is the 3×2 configuration w/ Stage 0, which is also the configuration used in the original Coconut paper and in the main comparison below. Here, 3×2 denotes 3 latent curriculum stages with 2 CTs added at each stage. Under the same training budget, RiM consistently outperforms all Coconut variants.

Table 1: **Main Results.** Accuracy (↑) on GSM8K and GSM-Hard, with checkpoints selected using the cross-validation protocol on GSM8K. Values report mean ± standard error over 16 split repeats.

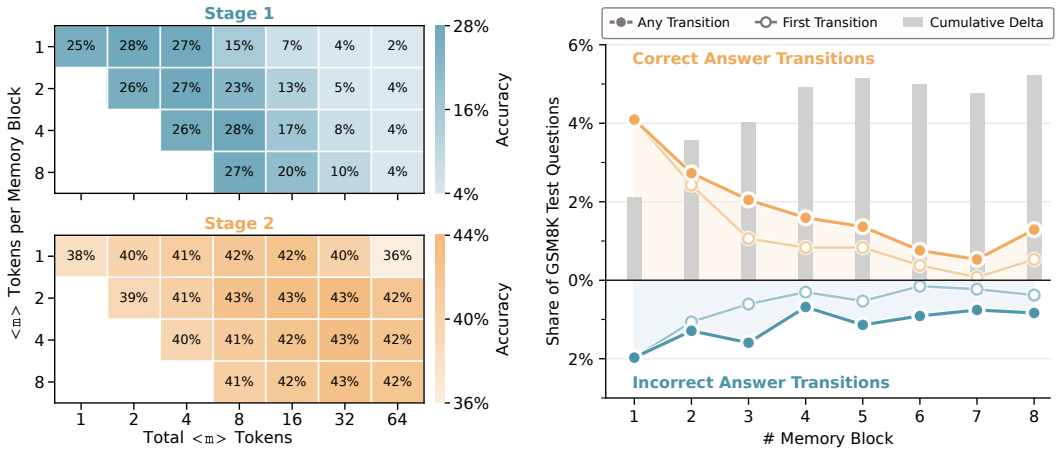
Model	Method	Variant	TTFT (ms)	GSM8K (ID)		GSM-Hard (OOD)	
				Greedy (%)	Pass@8 (%)	Greedy (%)	Pass@8 (%)
GPT-2	SFT	w/o CoT	**7.6**	15.4 ± 0.2	33.3 ± 0.3	3.5 ± 0.1	7.6 ± 0.1
	Coconut	w/ Stage 0	53.4	31.1 ± 0.2	45.0 ± 0.2	7.1 ± 0.0	10.7 ± 0.1
	RiM (ours)	Final block	**7.6**	**33.6** ± 0.2	**49.1** ± 0.2	**7.8** ± 0.1	**11.2** ± 0.1
Llama-3.2-1B	SFT	w/o CoT	**16.1**	23.9 ± 0.2	41.7 ± 0.3	5.3 ± 0.1	9.5 ± 0.1
	Coconut	w/ Stage 0	108.3	36.9 ± 0.2	51.1 ± 0.2	8.5 ± 0.0	12.2 ± 0.0
	RiM (ours)	Final block	**16.1**	**42.1** ± 0.2	**56.1** ± 0.3	**10.5** ± 0.0	**13.8** ± 0.0
Llama-3.2-3B	SFT	w/o CoT	**27.9**	36.2 ± 0.2	45.9 ± 0.2	8.5 ± 0.1	10.8 ± 0.1
	Coconut	w/ Stage 0	188.8	41.3 ± 0.2	55.5 ± 0.5	10.2 ± 0.1	13.5 ± 0.1
	RiM (ours)	Final block	**27.9**	**48.8** ± 0.2	**58.8** ± 0.2	**12.0** ± 0.0	**14.1** ± 0.0

Main results. Table 1 shows that the deployable final-block readout of RiM consistently outperforms the strongest Coconut variant, with improvements between 2.5 and 7.5 percentage points (pp) on GSM8K and between 0.7 and 1.8 pp on GSM-Hard. The gains over direct-answer SFT are even larger, with improvements between 12.6 and 18.2 pp on GSM8K and between 3.5 and 5.2 pp on GSM-Hard. The pass@8 accuracies, obtained by sampling 8 answers with temperature 1, show the same pattern. RiM samples correct solutions more reliably than the baselines across all benchmarks. This shows that memory blocks do not merely introduce stochastic variation but create useful internal computation that assigns substantial probability mass to correct answers, even when the greedy output is not always correct. Additionally, in Table 2, we compare RiM to SFT w/ CoT as a diagnostic reference for explicit reasoning. The results show that RiM often reaches correct solutions along its latent trajectory, approaching or even exceeding explicit CoT at substantially lower inference latency. Since any-block accuracy assumes access to the best memory-block readout, we treat it as a measure of the performance potential of RiM rather than as the main deployable result. Nonetheless, in Table 5, we show that linear probes on the memory block representations largely close the gap between final-block and any-block accuracy.

Inference latency. In Tables 1 and 2, we also report the time to first token (TTFT) per sample in milliseconds (ms), averaged over all GSM8K test questions and four independent runs. It can be observed that RiM has essentially the same TTFT as SFT w/o CoT. This is because the memory blocks are a fixed, small number of input tokens, processed in a single forward pass. By contrast, Coconut is about 7 times slower than RiM due to its autoregressive nature. SFT w/ CoT is about 27 times slower than RiM. These slowdowns are consistent across model scales. Thus, RiM has a favorable accuracy-latency tradeoff, improving substantially over direct-answer SFT and the strongest Coconut baseline while preserving direct-answer inference speed.

Table 2: **Additional Results.** Accuracy (↑) on GSM8K and GSM-Hard for SFT w/ CoT and the any-block readout of RiM. Checkpoints are selected using the same cross-validation protocol on GSM8K as in the main results. Values again report mean ± standard error over 16 split repeats.

Model	Method	Variant	TTFT (ms)	GSM8K (ID)		GSM-Hard (OOD)	
				Greedy (%)	Pass@8 (%)	Greedy (%)	Pass@8 (%)
GPT-2	SFT	w/ CoT	213.7	**39.8** ± 0.2	57.0 ± 0.2	8.4 ± 0.0	12.9 ± 0.1
	RiM (ours)	Any block	**7.6**	39.5 ± 0.3	**78.1** ± 0.2	**9.4** ± 0.0	**19.0** ± 0.1
Llama-3.2-1B	SFT	w/ CoT	420.3	49.1 ± 0.4	64.7 ± 0.3	11.2 ± 0.1	15.3 ± 0.1
	RiM (ours)	Any block	**16.1**	**51.4** ± 0.2	**76.8** ± 0.1	**13.0** ± 0.0	**19.6** ± 0.1
Llama-3.2-3B	SFT	w/ CoT	754.4	**66.9** ± 0.2	**78.3** ± 0.4	**19.0** ± 0.1	**24.4** ± 0.3
	RiM (ours)	Any block	**27.9**	57.3 ± 0.1	71.8 ± 0.2	13.8 ± 0.1	18.2 ± 0.0



(a) Accuracy across inference-time budgets.

(b) Answer transitions across memory blocks.

Figure 6: **Robustness across memory budgets.** RiM-trained Llama-3.2-1B on GSM8K test questions. *(a)* Greedy accuracy for different numbers and sizes of memory blocks after both Stage 1 and Stage 2. *(b)* Answer transition per memory block after Stage 2. Positive and negative values denote incorrect-to-correct and correct-to-incorrect changes, while gray bars show the cumulative net accuracy change.

4.3 RiM Maintains Accuracy Across Inference-Time Memory Budgets

Finally, we examine whether RiM maintains strong performance across inference-time memory budgets and how final answers evolve as more memory blocks are provided.

Performance across memory budgets. Figure 6a evaluates Stage 1 and Stage 2 checkpoints while varying the number of <m> tokens per memory block (M) on the y-axis and the number of memory blocks (K) on the x-axis. After Stage 1, greedy accuracy reaches around 27% on GSM8K for the first memory-block readouts, but degrades when K increases. This is expected, since Stage 1 ties memory blocks to intermediate reasoning steps rather than directly to the final answer. After Stage 2, however, this dependence largely disappears. Accuracy increases to about 43% and remains stable across a wide range of memory budgets. This suggests that Stage 2 converts the grounded memory blocks learned in Stage 1 into a fixed sequence of latent computations that can be read out reliably across positions, making RiM-trained models easy to deploy in practice.

Answer transitions. We next track answer transitions across memory blocks to determine whether additional memory blocks continue to refine the final answer or whether the readouts collapse to the same prediction. Figure 6b shows that, across memory blocks, final answers of the final model still change for a nontrivial fraction of questions, with a positive cumulative net effect. This suggests that the latent workspace does not collapse in Stage 2, but continues to refine predictions across memory blocks. It shows that RiM learns to use working memory in a stable yet nontrivial way, where additional memory blocks can change the answer to improve rather than destabilize performance. Further experimental details are provided in Section D, and additional results are reported in Section E.

5 Conclusion

In this work, we introduce Reasoning in Memory (RiM), showing that large language models can reason in working memory rather than having to generate intermediate thoughts autoregressively. Our method replaces reasoning steps with fixed memory blocks whose contextual representations form a latent workspace for task-specific computation. Since these memory blocks are fixed, they decouple internal reasoning from autoregressive generation and can be processed in a single forward pass.

Our experiments show that the two-stage curriculum turns the latent workspace induced by the memory blocks into a structured, block-specific, and sample-dependent workspace. This latent workspace improves performance over established baselines, remains robust across inference-time memory budgets, and preserves direct-answer inference speed. Future work could study whether reinforcement learning with final-answer rewards in Stage 2 can further improve the latent workspace, and explore hybrid approaches that combine RiM with explicit generation for more complex problems.

References

Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15619–15629, 2023.

Alan Baddeley. Working memory. *Science*, 255(5044):556–559, 1992.

Bradley C. A. Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv*, 2407.21787, 2024.

Gavin C. Cawley and Nicola L. C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11:2079–2107, 2010.

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

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*, 2110.14168, 2021.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv*, 2501.12948, 2025.

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

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

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, and Aurelien Rodriguez et al. The llama 3 herd of models. *arXiv*, 2407.21783, 2024.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: program-aided language models. In *International Conference on Machine Learning*, 2023.

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 *International Conference on Learning Representations*, 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*, 2412.06769, 2025.

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.

Nan Jiang, Ziming Wu, De-Chuan Zhan, Fuming Lai, and Shaobing Lian. DART: distilling autoregressive reasoning to silent thought. In *Conference on Empirical Methods in Natural Language Processing*, pages 5100–5108. Association for Computational Linguistics, 2025.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. *arXiv*, 2510.04871, 2025.

Eunki Kim, Sangryul Kim, and James Thorne. Learning to insert [PAUSE] tokens for better reasoning. In *Findings of the Association for Computational Linguistics: ACL*, Findings of ACL, pages 23760–23777. Association for Computational Linguistics, 2025.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, Kamile Lukosiute, Karina Nguyen, Newton Cheng, Nicholas Joseph, Nicholas Schiefer, Oliver Rausch, Robin Larson, Sam McCandlish, Sandipan Kundu, Saurav Kadavath, Shannon Yang, Thomas Henighan, Timothy Maxwell, Timothy Telleen-Lawton, Tristan Hume, Zac Hatfield-Dodds, Jared Kaplan, Jan Brauner, Samuel R. Bowman, and Ethan Perez. Measuring faithfulness in chain-of-thought reasoning. *arXiv*, 2307.13702, 2023.

Yann LeCun. A path towards autonomous machine intelligence. *OpenReview*, 2022.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024.

Jiayu Liu, Zhenya Huang, Anya Sims, Enhong Chen, Yee Whye Teh, and Ning Miao. MARCOS: deep thinking by markov chain of continuous thoughts. *arXiv*, 2509.25020, 2025.

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

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

Jacob Pfau, William Merrill, and Samuel R. Bowman. Let's think dot by dot: Hidden computation in transformer language models. *arXiv*, 2404.15758, 2024.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2018.

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

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv*, 2408.03314, 2024.

Yuchang Sun, Yanxi Chen, Yaliang Li, and Bolin Ding. Enhancing latent computation in transformers with latent tokens. *arXiv*, 2505.12629, 2025.

Jihoon Tack, Jack Lanchantin, Jane Yu, Andrew Cohen, Ilia Kulikov, Janice Lan, Shibo Hao, Yuandong Tian, Jason Weston, and Xian Li. LLM pretraining with continuous concepts. *arXiv*, 2502.08524, 2025.

Lev S. Vygotsky. *Mind in Society: Development of Higher Psychological Processes*. Harvard University Press, Cambridge, MA, 1978.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi-Yadkori. Hierarchical reasoning model. *arXiv*, 2506.21734, 2025a.

Jianwei Wang, Ziming Wu, Fuming Lai, Shaobing Lian, and Ziqian Zeng. Synadapt: Learning adaptive reasoning in large language models via synthetic continuous chain-of-thought. *arXiv*, 2508.00574, 2025b.

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

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*, volume 35, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, 2023.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Jintian Zhang, Yuqi Zhu, Mengshu Sun, Yujie Luo, Shuofei Qiao, Lun Du, Da Zheng, Huajun Chen, and Ningyu Zhang. Lightthinker: Thinking step-by-step compression. In *Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2025.

A Acknowledgments

The ELLIS Unit Linz, the LIT AI Lab, the Institute for Machine Learning, are supported by the Federal State Upper Austria. We thank the projects FWF AIRI FG 9-N (10.55776/FG9), AI4GreenHeatingGrids (FFG- 899943), Stars4Waters (HORIZON-CL6-2021-CLIMATE-01-01), FWF Bilateral Artificial Intelligence (10.55776/COE12). We thank NXAI GmbH, Audi AG, Silicon Austria Labs (SAL), Merck Healthcare KGaA, GLS (Univ. Waterloo), TÜV Holding GmbH, Software Competence Center Hagenberg GmbH, dSPACE GmbH, TRUMPF SE + Co. KG. Lukas would also like to thank Stefan Arnreiter for his encouragement during this work in the spirit of NIPSILD.

B Broader Impact

This work studies latent reasoning methods for language models on mathematical reasoning benchmarks. A potential positive impact is improved inference efficiency for reasoning systems, since RiM aims to replace autoregressively generated intermediate thoughts with fixed memory blocks that can be processed in a single forward pass. More efficient reasoning could reduce computational cost and energy use for applications that require multi-step problem solving. At the same time, moving intermediate computation into latent memory blocks can make the reasoning process less directly interpretable than written chain-of-thought traces. Our experiments do not release a new pretrained model or dataset, and they are limited to existing mathematical reasoning benchmarks.

C Assets and Licenses

We use existing public datasets, pretrained model families, and baseline implementations, all of which are cited in the main text. The GSM8K dataset [Cobbe et al., 2021] is listed under the MIT License.[1] GSM-Hard [Gao et al., 2023] is listed under the MIT License.[2] For GSM8K-Aug [Deng et al., 2023], our experiments use the `zen-E/GSM8k-Aug` Hugging Face distribution, which is listed under Apache-2.0.[3] Regarding pretrained language models, Llama-3.2-1B and Llama-3.2-3B [Dubey et al., 2024] are governed by the Llama 3.2 Community License and Acceptable Use Policy,[4,5] while GPT-2 [Radford et al., 2018] is released by OpenAI under a Modified MIT License[6]. Our Coconut baseline implementation is based on the official Coconut codebase [Hao et al., 2025], which is released under the MIT License.[7]

D Experimental Details

All experiments were run on one node with 8 NVIDIA H200-SXM-144GB GPUs.

Dataset Details. Table 3 summarizes the reasoning-step distribution of GSM8K-Aug, which determines the maximum Stage 1 memory-block depth and the number of update steps per epoch. The distribution is concentrated on short traces, but includes samples with up to 13 reasoning steps. Figure 7 shows representative samples from the training and out-of-distribution evaluation datasets. For GSM8K-Aug, we use the explicit arithmetic reasoning steps for Stage 1 supervision, whereas for GSM-Hard, we only use the question and final target answer at evaluation time.

Baselines. We compare against supervised fine-tuning (SFT) baselines, Coconut [Hao et al., 2025] as the explicit latent reasoning method, and the official DART numbers [Jiang et al., 2025]:

- **SFT w/o CoT.** The model is trained and evaluated directly from question to answer. This is the closest non-latent control for our forced-answer setting, since, like RiM, it cannot write explicit reasoning traces at test time.
- **SFT w/ CoT.** The model is trained and evaluated with full natural-language reasoning traces. This is not computationally equivalent to RiM, because it externalizes intermediate reasoning in text, but it provides the gold-standard reference for performance when explicit reasoning is allowed.

[1] https://huggingface.co/datasets/openai/gsm8k
[2] https://huggingface.co/datasets/reasoning-machines/gsm-hard
[3] https://huggingface.co/datasets/zen-E/GSM8k-Aug
[4] https://huggingface.co/meta-llama/Llama-3.2-1B
[5] https://huggingface.co/meta-llama/Llama-3.2-3B
[6] https://github.com/openai/gpt-2
[7] https://github.com/facebookresearch/coconut

Table 3: **GSM8K-Aug composition.** Distribution of training samples by number of reasoning steps.

Reasoning Steps	Training Samples	Share
1	62,908	16 %
2	143,578	37 %
3	104,249	27 %
4	48,198	13 %
5	17,906	5 %
6	5,666	1 %
7	2,359	1 %
8	577	0 %
9	126	0 %
10	43	0 %
11	8	0 %
12	1	0 %
13	1	0 %
Total	**385,620**	

- **Coconut w/o Stage 0.** Hao et al. [2025] replace the explicit CoT with continuous thoughts (CTs), feeding previous hidden states back as the next input embedding instead of decoding them into word tokens. The Coconut curriculum progressively removes early written reasoning steps and inserts CTs, while training the model to predict the remaining reasoning trace and final answer. This variant omits Stage 0, the initial training phase on explicit reasoning traces equivalent to SFT w/ CoT. It therefore starts directly from the latent curriculum, making it the most direct latent reasoning comparison to RiM at the Llama model scales.
- **Coconut w/ Stage 0.** This variant follows the original Coconut recipe more closely by adding Stage 0 and therefore using a larger effective training budget. We compare against this stronger variant in our main results in Table 1 across all model scales. Our Coconut implementation is based on the official codebase [Hao et al., 2025].
- **DART.** We attempted to reimplement DART, but to the best of our knowledge, no official codebase has been made public to date, and we could not obtain a reliable reproduction. We therefore compare to DART using the official results reported by Jiang et al. [2025] in Table 6. This comparison is conservative for RiM in terms of training cost, since the official DART results train Llama-3.2-1B and Llama-3.2-3B for 10 epochs each, and GPT-2 for 40 epochs. Since DART also requires two training pathways, this corresponds to a significantly higher training cost than RiM. Nevertheless, RiM outperforms the reported DART results in all comparable settings (see Table 6).

RiM. We use 2 <m> tokens per memory block, matching Coconut, which uses 2 additional CTs at each stage. Since GSM8K-Aug contains samples with up to 13 reasoning steps (see Table 3), Stage 1 uses up to 13 memory blocks, one for each reasoning step. In Stage 2, the reasoning-step targets are discarded, and only the final answer is supervised, so the number of memory blocks no longer has to match the number of reasoning steps. We therefore fix the number of memory blocks to 8, yielding 32 special tokens in total when including the block-boundary tokens. We choose this budget to align with Jiang et al. [2025], who find that 30 silent tokens are optimal for their method, enabling a fair comparison. We train for 6 epochs in Stage 1, corresponding to 18,000 training steps, and 2 epochs in Stage 2, corresponding to 6,000 training steps. For GPT-2, we adopt the same initial Stage 0 used for Coconut w/ Stage 0. For the Llama models, we start directly from the base checkpoints without Stage 0.

Hyperparameters. We train all models with rank-128 LoRA adapters using bfloat16 precision [Hu et al., 2022]. For all methods, we use a global batch size of 128, resulting in about 3,000 update steps per epoch on GSM8K-Aug [Deng et al., 2023]. This training setup largely follows prior work on latent reasoning [Hao et al., 2025, Jiang et al., 2025, Shen et al., 2025]. We use a constant learning-rate schedule with a 4% warmup period and a weight decay of 0.01 by default. In initial experiments, we found that training behavior was largely insensitive to the number of warmup steps and to moderate changes in weight decay. We therefore keep these parameters fixed across all experiments. In contrast, we found the learning rate to have a substantial impact on training behavior. Thus, we perform a separate learning-rate search for each method-model combination over the range $[10^{-5}, 10^{-3}]$ for a fair comparison. For Coconut, we additionally search over the maximum number of stages on Llama-3.2-1B and use the best-performing variant for all main comparisons (see Figure 5b).

Question:

John begins his hike at 8:30 AM and finishes at 6:30 PM. He rests for 20 minutes at noon, 15 minutes in the afternoon and 30 minutes before finishing his hike. How many hours did he spend hiking?

Reasoning Steps:

$$\begin{aligned} \text{«}18.5 - 8.5 &= 10\text{»} \\ \text{«}20 + 15 + 30 &= 65\text{»} \\ \text{«}65/60 &= 1.08333\text{»} \\ \text{«}10 - 1.08333 &= 8.91667\text{»} \end{aligned}$$

Answer:

The final answer is $\boxed{8.92}$

(a) GSM8K-Aug training sample with a question, four reasoning steps, and the final target answer.

Question:

Mrs. Cruz is looking for a house that will not go beyond her $400 000 budget. She saw a property that has a selling price of $350 000. On top of that, the buyer has to pay a brokerage fee which is 5% of the selling price, and also the transfer fee that is 12% of the selling price. How much more is the total price of the house than Mrs. Cruz's budget?

Answer:

The final answer is $\boxed{9500.0}$

(b) GSM-Hard test sample with a question and the final target answer.

Figure 7: **Dataset samples.** Random samples from the training and test datasets that were used throughout our experiments.

Custom Attention Mask. We train with a custom block-causal attention mask that separates the sequence into a memory block stream and supervised written reasoning step readout branches. Future memory blocks may attend to the question and previous memory blocks, but never to supervised reasoning step readouts. Readouts may attend to the question and the memory blocks available up to their position, but not to other readouts. This allows all reasoning steps or answer targets to receive teacher-forced supervision in a single forward pass while preventing later latent states from reading the ground-truth reasoning step. In the default setting used throughout our experiments, tokens within each memory block remain causally ordered. However, one can also consider the variant shown in Figure 8, where tokens within a memory block attend bidirectionally to each other. This increases within-block communication while preserving the same block-level causal structure. Empirically, however, the bidirectional variant yields mixed results, with no consistent trend across models or benchmarks. We therefore leave a systematic study of within-block attention structure to future work.

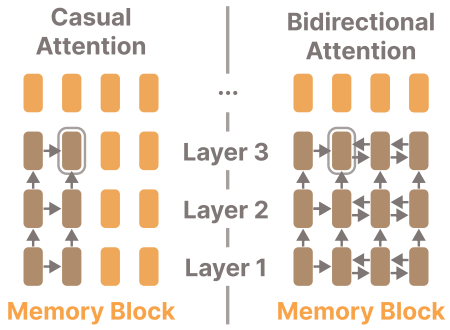
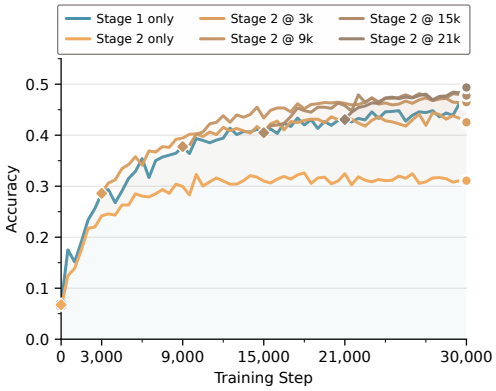
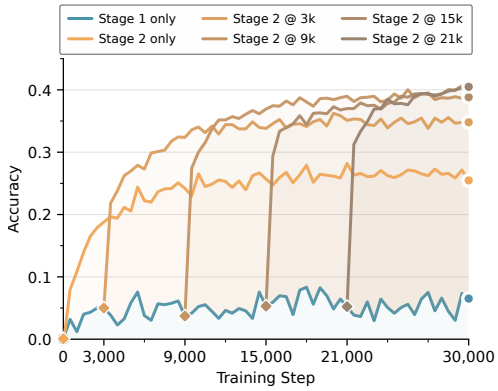


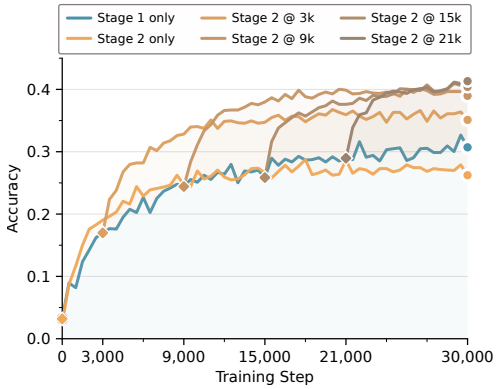
Figure 8: **Bidirectional memory block attention.** In this variant, tokens inside the same memory block can attend to each other bidirectionally, while attention between memory blocks remains causal.



(a) Any-block accuracy.



(b) Final-block accuracy.



(c) Best-block accuracy.

Figure 9: **Stage-switch ablation.** GSM8K evaluation accuracy for Llama-3.2-1B trained on GSM8K-Aug, varying the stage switch from Stage 1 to Stage 2.

E Additional Results

Stage-switch ablation. Figure 9 ablates the transition from Stage 1 to Stage 2. Training with Stage 2 alone improves final-answer accuracy quickly, but plateaus well below runs that first ground the memory blocks with Stage 1. This supports the claim that dense subversion signal is important. Conversely, Stage 1 alone produces high any-block accuracy, but its final-block accuracy remains low because the model has not been trained to use a fixed final readout. This supports the claim that switching to Stage 2 after sufficient Stage 1 training is important. As a result, the latent workspace learned in Stage 1 is converted into stronger final-block and best-block accuracy. Very early switches are weaker, indicating that the memory blocks must first acquire a useful computational role before answer-only refinement becomes effective.

RiM vs. Coconut curriculum. To isolate the effect of our two-stage curriculum as presented in Section 3, we compare it with the staged curriculum used for Coconut Hao et al. [2025], which was inspired by Deng et al. [2024]. This ablation keeps the fixed memory blocks of RiM, but replaces our dense supervision signal with the gradual Coconut-style curriculum. Figure 10 shows that the Coconut curriculum remains substantially below the RiM curriculum across training steps. This confirms that memory blocks require a dense supervision signal that forces intermediate computation through the latent workspace in order to achieve high downstream performance.

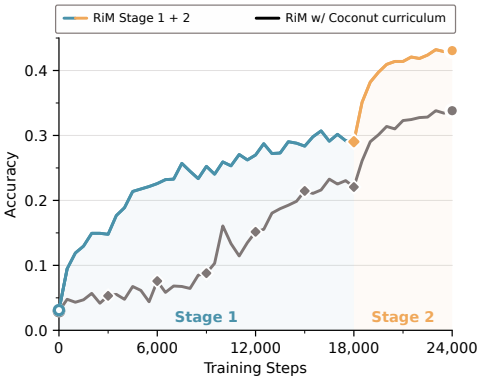
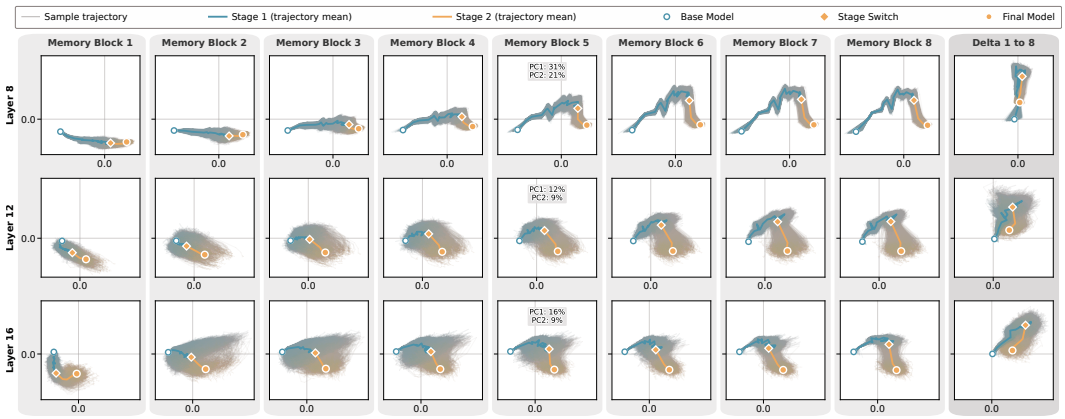
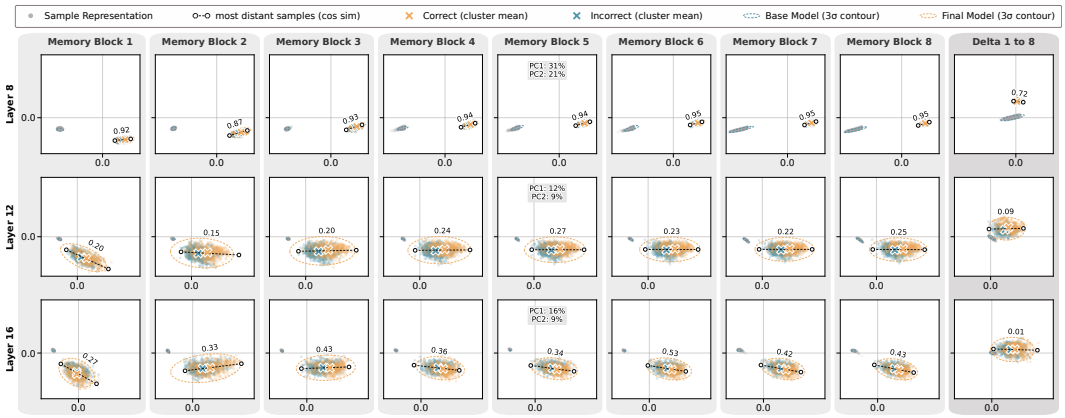


Figure 10: **RiM vs. Coconut curriculum.**



(a) Training Trajectories.



(b) Initial and Final Representations.

Figure 11: **Memory block representations.** Using all GSM8K test questions, we project memory block representations and the first-to-final memory block representation delta into a shared PCA basis. The top row shows their trajectories during training. The bottom row shows the representations from the initial base model (Llama-3.2-1B) and the final RiM-trained model in the same PCA basis.

Latent computation in memory blocks. Figure 11 expands the representation analysis from Figure 4 of the main paper. For each layer, we collect the contextual representation of the last <m> token in each memory block across all GSM8K test questions and project them from both stages into a shared PCA basis. The same basis is used for the first-to-final memory block representation delta, so panels within each row are directly comparable.

The training trajectories in Figure 11a show that memory block representations move along smooth and block-specific trajectories. Stage 1 already organizes the latent workspace, while Stage 2 changes the direction of the trajectories rather than collapsing them. This matches the objective shift, as Stage 1 grounds the memory blocks with reasoning-step supervision, whereas Stage 2 removes this strict step-level supervision and converts the grounded memory blocks into a fixed sequence of latent computations for final-answer prediction.

Figure 11b compares the initial base model with the final RiM model in the same PCA bases. The base-model representations are largely collapsed, while the final representations form broad, sample-dependent clouds whose locations remain organized by memory block. The first-to-final delta makes this especially clear, as in later layers, the delta representations become highly diverse, with the annotated most-distant-sample cosine similarity dropping to 0.09 in Layer 12 and 0.01 in Layer 16. Thus, the memory blocks are not used as fixed placeholders but become structured, block-specific, and sample-dependent latent states. The figures provide representation-level evidence that RiM trains the model to use memory blocks as a latent workspace for task-relevant intermediate computation.

Table 4: **Main Results.** Accuracy (↑) on two evaluation benchmarks, with checkpoints selected using the cross-validation protocol on GSM8K. Values report mean ± standard error over 16 split repeats. Bold indicates the best result, and underline indicates the second-best result under each model.

Model	Method	Variant	TTFT (ms)	GSM8K (ID)		GSM-Hard (OOD)	
				Greedy (%)	Pass@8 (%)	Greedy (%)	Pass@8 (%)
GPT-2	SFT	w/o CoT	**7.6**	15.4 ± 0.2	33.3 ± 0.3	3.5 ± 0.1	7.6 ± 0.1
		w/ CoT	213.7	**39.8** ± 0.2	<u>57.0</u> ± 0.2	<u>8.4</u> ± 0.0	<u>12.9</u> ± 0.1
	Coconut	w/o Stage 0	53.4	22.3 ± 0.3	40.6 ± 0.3	5.2 ± 0.0	8.9 ± 0.2
		w/ Stage 0		31.1 ± 0.2	45.0 ± 0.2	7.1 ± 0.0	10.7 ± 0.1
	RiM (ours)	Final block	**7.6**	33.6 ± 0.2	49.1 ± 0.2	7.8 ± 0.1	11.2 ± 0.1
		Any block		<u>39.5</u> ± 0.3	**78.1** ± 0.2	**9.4** ± 0.0	**19.0** ± 0.1
Llama-3.2-1B	SFT	w/o CoT	**16.1**	23.9 ± 0.2	41.7 ± 0.3	5.3 ± 0.1	9.5 ± 0.1
		w/ CoT	420.3	<u>49.1</u> ± 0.4	<u>64.7</u> ± 0.3	<u>11.2</u> ± 0.1	<u>15.3</u> ± 0.1
	Coconut	w/o Stage 0	108.3	30.1 ± 0.3	46.1 ± 0.3	6.8 ± 0.1	11.0 ± 0.1
		w/ Stage 0		36.9 ± 0.2	51.1 ± 0.2	8.5 ± 0.0	12.2 ± 0.0
	RiM (ours)	Final block	**16.1**	42.1 ± 0.2	56.1 ± 0.3	10.5 ± 0.0	13.8 ± 0.0
		Any block		**51.4** ± 0.2	**76.8** ± 0.1	**13.0** ± 0.0	**19.6** ± 0.1
Llama-3.2-3B	SFT	w/o CoT	**27.9**	36.2 ± 0.2	45.9 ± 0.2	8.5 ± 0.1	10.8 ± 0.1
		w/ CoT	754.4	**66.9** ± 0.2	**78.3** ± 0.4	**19.0** ± 0.1	**24.4** ± 0.3
	Coconut	w/o Stage 0	188.8	42.2 ± 0.2	56.8 ± 0.4	10.1 ± 0.0	13.3 ± 0.1
		w/ Stage 0		41.3 ± 0.2	55.5 ± 0.5	10.2 ± 0.1	13.5 ± 0.1
	RiM (ours)	Final block	**27.9**	48.8 ± 0.2	58.8 ± 0.2	12.0 ± 0.0	14.1 ± 0.0
		Any block		<u>57.3</u> ± 0.1	<u>71.8</u> ± 0.2	<u>13.8</u> ± 0.1	<u>18.2</u> ± 0.0

Detailed main results. Table 4 expands the main results from Table 1 by reporting all baseline variants and an additional ablation for RiM. The final-block row corresponds to the deployable setting used in the main paper and remains the strongest answer-only method across all backbones and benchmarks. The any-block row asks whether any memory-block readout produces the correct answer, and therefore cannot be interpreted as a deployable selection rule. Its large gains, especially for pass@8, show that the sequence of memory-block readouts often contains correct solutions even when the final-block readout is incorrect under deterministic decoding. This suggests that RiM does more than improve a single readout, but instead creates a sequence of latent states from which useful candidate solutions become available at different depths.

Probe-based answer selection. We next ask whether the memory-block representations analyzed in Figure 11 contain enough information to identify the correct readouts. For each memory block, we train a lightweight linear probe on the corresponding memory-block representations from 256 held-out GSM8K samples to predict whether the readout after that memory block is correct. For an unseen evaluation sample, we group memory blocks whose generated answers are equivalent and combine the calibrated probe probabilities within each group, assigning higher confidence to answers supported by multiple blocks or by a highly confident block. We then select the answer group with the

Table 5: Linear probe separability and answer-selection accuracy by memory block. Values are reported in percent, and standard errors are percentage points computed over 16 split repeats.

Metric	Memory block					Probe-based Answer Selection
	1	2	4	6	8	
AUROC (↑)	84.8 ± 0.1	85.0 ± 0.1	84.2 ± 0.1	83.6 ± 0.1	84.5 ± 0.1	**86.0** ± 0.1
AUPRC (↑)	80.7 ± 0.2	82.3 ± 0.2	82.0 ± 0.2	81.6 ± 0.2	81.9 ± 0.2	**83.3** ± 0.2
Accuracy (↑)						**90.0** ± 0.2

Table 6: **Official numbers.** Greedy accuracy on GSM8K and GSM-Hard, taking the official numbers from prior work. *official numbers from Jiang et al. [2025], †official numbers from Hao et al. [2025].

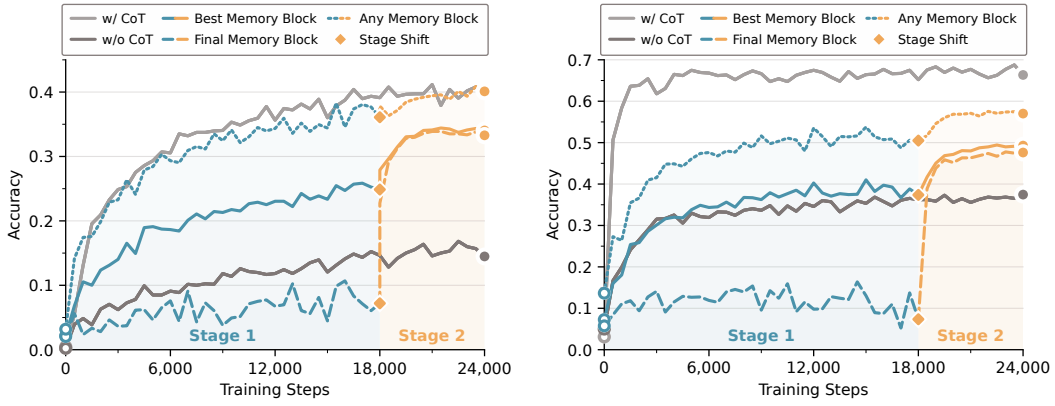
Model	GSM8K (ID)			GSM-Hard (OOD)		
	DART	Coconut	RiM (ours)	DART	Coconut	RiM (ours)
GPT-2	24.7*	34.1†	**35.5**	–	–	8.4
Llama-3.2-1B	42.6*	**50.6***	43.1	10.9*	**11.2***	**11.2**
Llama-3.2-3B	46.6*	43.6†	**49.4**	–	–	12.1

highest combined confidence. In Table 5, we report AUROC and AUPRC for these selected-answer confidence scores across 16 held-out splits, showing that correctness is highly predictable from the memory-block representations. Conditioned on the recoverable subset where at least one memory block produces a correct answer, this selection procedure chooses a correct answer 90% of the time. This suggests that much of the gap between final-block and any-block accuracy can be closed with simple linear probes. These results provide a simple proof of concept that correctness information is accessible from the memory-block representations, while more involved selection mechanisms are left to future work.

Comparison with Official Numbers. Official results from prior latent reasoning baselines are not fully controlled comparisons, as methods differ in their training setups and evaluation protocols. We therefore treat Table 6 as a contextual comparison to the literature rather than as the primary evidence for our method. For completeness, we report RiM under the same convention used for the official numbers. Instead of using the held-out cross-validation protocol from Table 1, we select the best checkpoint on the full evaluation set, allowing for a more direct comparison to the official numbers.

Even under this convention, the comparison remains conservative with respect to training cost. Under the Coconut method, GPT-2 is trained for 18 epochs [Hao et al., 2025], Llama-3.2-1B for 10 epochs [Jiang et al., 2025], and Llama-3.2-3B for 8 epochs [Hao et al., 2025]. Moreover, the models are fine-tuned rather than trained with parameter-efficient LoRA adapters, which makes training substantially more expensive. Relative to our setup of 8 epochs with LoRA adapters, the number of training epochs alone corresponds to roughly 2.25× higher training cost for GPT-2 and 1.25× higher training cost for Llama-3.2-1B, before accounting for the additional cost of full-model fine-tuning.

The comparison to DART is even more conservative. Under the DART method, Llama-3.2-1B and Llama-3.2-3B are trained for 10 epochs each, and GPT-2 is trained for 40 epochs [Jiang et al., 2025]. In addition, DART uses two separate training pathways that require two forward passes per sample and optimizes three loss terms simultaneously. Relative to our single-forward-pass training for 8 epochs, this corresponds to roughly 2.5× higher training cost for the Llama models and 10× higher training cost for GPT-2.



(a) GPT-2 (b) Llama-3.2-3B

Figure 12: **GSM8K training curves.** Greedy accuracy on GSM8K test questions over training, comparing RiM to SFT when training GPT-2 and Llama-3.2-3B on GSM8K-Aug.

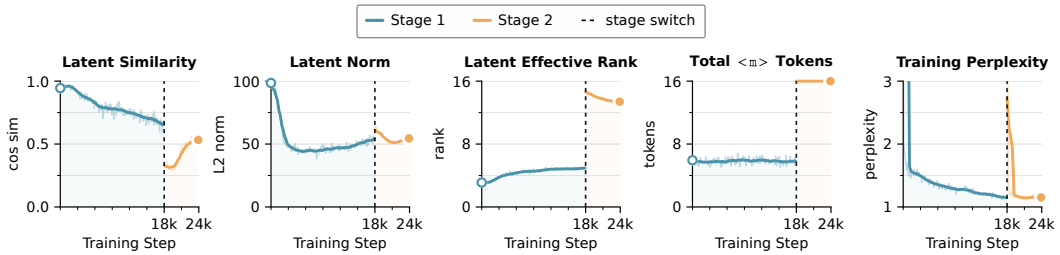


Figure 13: **Training diagnostics for RiM.** From left to right: average within-block latent cosine similarity, average latent-state norm, latent effective rank, the active <m> token budget, and student training perplexity. The dashed line marks the transition from Stage 1 to Stage 2.

Despite this larger training budget, RiM outperforms the reported DART results in all comparable settings. More broadly, our method is best or tied for best in three of the four settings for which official DART or Coconut numbers are available. Thus, while differences in training setup preclude a strictly controlled evaluation, this official-number comparison shows that RiM is highly competitive with substantially more involved latent reasoning methods.

Training curves across model scales. Figure 12 complements the main training curves in Figure 5, comparing RiM with SFT using GPT-2 and Llama-3.2-3B. The any-block accuracy rises substantially above direct-answer SFT, indicating that correct answers appear at some memory depths before the model is explicitly trained to use a fixed final block. After the stage switch, Stage 2 rapidly transfers this latent computation into the deployable final-block readout, which improves over direct-answer SFT while retaining fixed answer-only inference latency. The gap to explicit CoT remains larger for Llama-3.2-3B, but this baseline also spends additional decoding steps in language space and is therefore not latency-matched to RiM.

Training metrics. The geometric metrics in Figure 13 are computed from the final hidden layer of the latent tokens every 100 optimizer steps and averaged over distributed workers. We measure within-block latent similarity as the average pairwise cosine similarity between latent-token hidden states within the same contiguous memory block. This diagnostic captures whether tokens inside a memory block collapse to a common state or specialize into distinct directions. We also track the latent norm, computed as the average Euclidean norm of the latent-token hidden states. To measure the dimensionality of the latent workspace, we compute the effective rank from the singular-value spectrum of the matrix containing all latent-token states for each sample. Higher values indicate that the latent-token subspace uses more independent directions. We additionally log adjacent latent-state similarity, defined as the cosine similarity between consecutive latent tokens, as a smoothness diagnostic. Finally, student perplexity is the exponentiated token-averaged student NLL on the supervised training targets.

Empirical latency. Table 7 additionally reports the latency for generating the full answer, complementing the time to first token (TTFT). The table records generated answer length and wall-clock time per GSM8K test question. RiM has essentially the same measured latency as SFT w/o CoT as the memory blocks are fixed input tokens processed in a single forward pass. By contrast, SFT w/ CoT and Coconut are substantially slower because they require autoregressive generation of intermediate text tokens or CTs. This makes the inference-cost comparison explicit and shows that RiM preserves direct-answer inference speed.

Table 7: **Inference latency.** Average tokens generated and wall-clock time in milliseconds (ms) per GSM8K question with Llama-3.2-1B, with standard errors across 4 runs.

Method	Tokens	ms	Δ ms
SFT w/o CoT	3.1	126.0 $\pm$ 0.5	—
SFT w/ CoT	36.7	1108.7 $\pm$ 3.0	+982.7
Coconut	3.1	304.7 $\pm$ 0.9	+178.7
RiM (ours)	3.1	**126.5** $\pm$ 0.5	**+0.5**