CAN LLM AGENTS COMMUNICATE WITHOUT AUTOREGRESSION? THOUGHT BLOCKS FOR LATENT MULTI-AGENT COLLABORATION

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-based multi-agent systems typically communicate through text, incurring repeated generation and re-encoding costs. Latent communication removes explicit verbalization, but constructing a latent message may still require a sequential latent trajectory. We therefore distinguish *message representation* from *message construction* and ask: can an agent directly construct a useful continuation state without executing the trajectory that normally produces it? We formulate this as trajectory-to-state compilation and instantiate it with Thought Blocks, fixed-size blocks of latent key–value (KV) memory constructed in one shot. Thought Blocks map a compact continuation representation and compressed contextual KV memory to parallel latent positions, which are rendered into model-native KV continuation states in a single frozen-backbone forward pass. We train the writer by distilling continuation states reached by an autoregressive latent teacher, whose trajectory is omitted at inference. Across multiple LLM backbones and reasoning and question-answering tasks, Thought Blocks substantially reduce sequential communication computation and latency relative to autoregressive latent communication while maintaining competitive accuracy, achieving a 2.5× average latency speedup. Compared with text-based multi-agent communication, the efficiency gains are broader still, reducing text-token traffic, autoregressive computation, and latency.

1 INTRODUCTION

LLM-based multi-agent systems have emerged as a promising paradigm for solving complex tasks through decomposition, specialization, and iterative refinement (Li et al., 2023; Wu et al., 2023; Hong et al., 2023; Qian et al., 2024; Du et al., 2023; Liu et al., 2023). As these systems are applied to increasingly long-horizon and computation-intensive workflows, communication between agents becomes an important systems concern. For example, OpenAI recently reported using a system of coordinating agents in its work on the Navier–Stokes Millennium Prize Problem, involving approximately 2.7 million inter-agent messages and 130 billion output tokens (OpenAI, 2026). At such scales, the cost of repeatedly constructing and exchanging intermediate information can become a substantial part of multi-agent computation.

Most existing multi-agent systems communicate through natural language (Qian et al., 2024; Yan et al., 2025). Recent latent-communication methods challenge this interface by showing that intermediate information can instead be carried through model-internal representations (Zou et al., 2025; Du et al., 2025; Ramesh & Li, 2025). More importantly, recent mechanistic evidence from the Jacobian Lens identifies a small internal "J-space" in which intermediate results can be retained and reused by subsequent computation (Gurnee et al., 2026). This suggests a separation between the trajectory that produces an intermediate result and the state through which it becomes available to later computation. For inter-agent communication, this motivates a natural hypothesis: a receiver may need only a functionally sufficient continuation state, rather than the sender's full latent computation trajectory.

This possibility highlights a key distinction: message representation and message construction are separate design choices, and a message can be latent while its construction remains autoregressive. For example, some latent multi-agent protocols eliminate textual messages but still generate a sequence of latent thoughts before transferring the resulting latent state to the next agent (Zou et al., 2025; Li et al.,

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.

2026). The communicated object is latent, yet constructing it still requires a sequential latent rollout. We therefore ask: *must a useful inter-agent latent state be obtained by executing the trajectory that normally produces it, or can it be constructed directly from the sender's current state?* We formulate this as *non-autoregressive latent communication (NALC)*: constructing inter-agent latent messages without a sequential latent rollout. We call the underlying principle *trajectory-to-state compilation*: directly constructing the continuation state that the rollout would otherwise produce.

We instantiate this idea with *Thought Blocks*, fixed-size blocks of latent KV continuation memory constructed in one shot. Our method represents the sender using a factorized continuation state consisting of a compact *continuation seed*, which indicates how computation should continue, and a *context bank*, which preserves contextual information that subsequent computation may need to access. A Thought Block writer maps this state directly to an outgoing KV block without autoregressive intermediate-message generation. To train the writer, we use an autoregressive latent teacher to identify useful continuation states and distill these state-level targets into the one-shot writer. The teacher trajectory is therefore used only for supervision during training and is removed entirely from inter-agent message construction at inference time.

Experiments across reasoning and question-answering benchmarks and multiple LLM backbones show that direct continuation-state construction provides a favorable accuracy–efficiency trade-off. Compared with text-based multi-agent communication, Thought Blocks reduce text tokens by 45.7%, autoregressive forward steps by 84.8%, and end-to-end latency by 8.6$\times$ on average, with an average accuracy difference of only -0.15 points. More importantly, compared with autoregressive latent communication, where both methods already communicate in latent space, Thought Blocks further reduce autoregressive forward steps by 35.7%, with an average accuracy difference of -0.72 points. These results provide empirical evidence that useful inter-agent continuation memory need not be constructed by replaying an autoregressive latent trajectory.Our contributions are threefold:

- **We formulate non-autoregressive latent communication as a message-construction problem.** We distinguish the representation of an inter-agent message from the process used to construct it, and introduce *trajectory-to-state compilation* as directly constructing a useful continuation state without executing the trajectory that would normally produce it.
- **We introduce Thought Blocks for one-shot latent communication.** Thought Blocks instantiate trajectory-to-state compilation by directly constructing fixed-size KV continuation memory from a factorized sender state and are trained via state-level distillation from autoregressive latent trajectories.
- **We show that autoregressive latent message construction can be substantially reduced while largely preserving downstream utility.** Across multiple backbones and diverse reasoning and question-answering tasks, Thought Blocks substantially reduce sequential communication computation, latency, and KV-memory load while maintaining competitive task performance.

2 Related Work

2.1 Efficient Multi-Agent Communication

LLM-based multi-agent frameworks typically coordinate specialized roles through natural-language interaction (Li et al., 2023; Wu et al., 2023; Hong et al., 2023; Qian et al., 2024). Work on communication efficiency reduces this textual overhead through learned communication policies (Chen et al., 2024) or by pruning redundant agents and links (Wang et al., 2025), while retaining natural language as the communication medium.

A complementary line of work communicates through model-internal representations. Activation Communication (AC) exchanges intermediate activations (Ramesh & Li, 2025); InterLat communicates hidden states and studies latent compression (Du et al., 2025); and Thought Communication extracts latent factors for collaboration (Zheng et al., 2025). At the KV-cache level, KVCOMM reuses and adapts KV states across contexts (Ye et al., 2025), KVComm selectively shares informative KV pairs (Shi et al., 2026), C2C projects and fuses source and receiver caches (Fu et al., 2025), and LCF learns compact cache channels (Rossi et al., 2026). StateBridge aligns sender hidden states to the receiver space without training (Peng et al., 2026), while recent work studies heterogeneous KV communication through dense cross-model cache alignment (Chen et al., 2026) and dual-cache

Table 1: Comparison of latent reasoning and communication mechanisms.

Method	Inter-agent communication	KV message	Sequential latent rollout	Iterative latent refinement	Direct post-trajectory state
Single-agent latent reasoning					
Coconut Hao et al. (2024)	✗	✗	✓	✗	✗
PCCoT Wu et al. (2025)	✗	✗	✗	✓	✗
CoT2 Gozeten et al. (2026)	✗	✗	✓	✗	✗
LaDiR Kang et al. (2025)	✗	✗	✗	✓	✗
Trajectory-produced latent communication					
InterLat-Comp. Du et al. (2025)	✓	✗	✓	✗	✗
LessLatent Li et al. (2026)	✓	✓	✓	✗	✗
LatentMAS Zou et al. (2025)	✓	✓	✓	✗	✗
Latent-state transfer / transformation					
AC Ramesh & Li (2025)	✓	✗	✗	✗	✗
InterLat Du et al. (2025)	✓	✗	✗	✗	✗
Thought Communication Zheng et al. (2025)	✓	✗	✗	✗	✗
KVCOMM (Ye et al.) Ye et al. (2025)	✓	✓	✗	✗	✗
KVComm (Shi et al.) Shi et al. (2026)	✓	✓	✗	✗	✗
C2C Fu et al. (2025)	✓	✓	✗	✗	✗
LCF Rossi et al. (2026)	✓	✓	✗	✗	✗
StateBridge Peng et al. (2026)	✓	✗	✗	✗	✗
XKV Liu et al. (2026)	✓	✓	✗	✗	✗
Thought Blocks (Ours)	✓	✓	✗	✗	✓

transformation (Liu et al., 2026). Relayed KV memories can also be compressed while preserving task-relevant information (Li et al., 2026).

These methods improve how model-internal states are selected, aligned, transferred, or compressed. In particular, cache-transfer methods primarily operate on sender states that have already been materialized. Our setting instead targets a continuation state that is not yet available at the communication boundary and would ordinarily be reached only after additional latent computation. We ask whether this post-trajectory state can be constructed directly from the pre-trajectory sender state, without executing that trajectory. Accordingly, we use *non-autoregressive latent communication* to refer specifically to eliminating trajectory-based message construction, rather than broadly to any latent protocol that avoids token-by-token text decoding.

2.2 Latent Trajectories and Direct State Construction

Latent representations do not by themselves eliminate sequential computation. Coconut recurrently feeds hidden states back as continuous inputs (Hao et al., 2024); LatentMAS autoregressively generates latent thoughts before relaying the resulting working memory (Zou et al., 2025); and InterLat's compressed variant likewise uses recurrent latent computation (Du et al., 2025). Thus, latent messages may still require a sequence of intermediate states to construct.

Other work reduces or reorganizes this sequential structure. PCCoT uses Jacobi iteration to update multiple continuous thought positions in parallel (Wu et al., 2025); CoT2 tracks multiple reasoning alternatives in parallel within each decoding step (Gozeten et al., 2026); and LaDiR constructs latent thought blocks through iterative diffusion-based denoising (Kang et al., 2025). These methods parallelize or iteratively refine latent computation, but inference still instantiates intermediate latent states before the resulting reasoning state becomes available.

We instead ask whether such a trajectory needs to be executed at all when its purpose is to produce a state for another agent. This motivates *trajectory-to-state compilation*: during training, an autoregressive teacher identifies a useful post-trajectory continuation state, while the student learns to map the pre-trajectory sender state directly to that target. At inference, the teacher rollout is omitted and the outgoing KV continuation memory is constructed in one shot. This separates our work from both transfer or transformation of already-materialized states and methods that accelerate latent trajectory construction. Table 1 summarizes this distinction: Thought Blocks perform inter-agent KV communication by directly constructing a post-trajectory continuation state, without sequential latent rollout or iterative refinement at inference.

3 METHOD

3.1 PROBLEM SETUP

We consider sequential collaboration among N language-model agents $\{A_i\}_{i=1}^{N}$. Given a task q, each agent A_i receives a local textual input x_i and, except for the first agent, an inter-agent message $\mathcal{M}_{i-1}$ from its predecessor, with $\mathcal{M}_0 = \varnothing$. The local input includes the agent's role instruction and may additionally contain the original task, retrieved evidence, or other observations.

Let $\mathcal{S}_i$ denote the sender-side state available after agent A_i has processed its local input and incoming message. A generic communication step can therefore be written as

$$\mathcal{S}_i = A_i(x_i, \mathcal{M}_{i-1}), \qquad \mathcal{M}_i = \Gamma_i(\mathcal{S}_i), \quad i < N, \tag{1}$$

where Γ_i denotes the mechanism that constructs the outgoing message from the sender state. The final agent consumes its local input and the last incoming message to produce the task output $\hat{y} = A_N(x_N, \mathcal{M}_{N-1})$. Different communication methods instantiate $\mathcal{M}_i$ and Γ_i differently. Text-based systems verbalize intermediate computation into natural-language tokens, whereas latent-communication systems expose model-internal states directly. In this work, we represent the communicated object as latent KV continuation memory and focus on a separate question: *how should this latent message be constructed?*

In trajectory-based latent communication, the outgoing state is obtained by first executing a sequential latent computation,

$$\tau_i = \mathcal{T}_{\theta^\star}(\mathcal{S}_i), \qquad \mathcal{M}_i^\star = \Psi_{\theta^\star}(\tau_i), \tag{2}$$

where $\tau_i = (u_{i,1}, \ldots, u_{i,T_{\text{lat}}})$ is a latent trajectory, $\mathcal{T}_{\theta^\star}$ denotes the sequential latent-computation process, and $\Psi_{\theta^\star}$ extracts the continuation memory communicated to the next agent. Although $\mathcal{M}_i^\star$ is latent, constructing it still requires executing the trajectory τ_i. Our goal is to replace this trajectory-dependent construction with a direct parameterized writer,

$$\mathcal{M}_i = \mathcal{W}_{\phi;\theta^\star}(\mathcal{S}_i), \tag{3}$$

where ϕ denotes trainable writer parameters and $\theta^\star$ denotes the frozen language-model backbone. We refer to this replacement of latent-trajectory execution with direct continuation-state construction as *trajectory-to-state compilation*.

The remainder of this section first specifies the continuation state $\mathcal{S}_i$, then describes how Thought Blocks directly construct an outgoing latent message from this state, and finally explains how this direct mapping is learned from autoregressive trajectories. At inference time, Eq. 3 is applied by each intermediate agent, while the final textual answer is generated through the standard autoregressive language-model interface.

3.2 FACTORIZED CONTINUATION STATE

Direct continuation construction requires preserving two distinct forms of information available after prefill: a boundary representation that conditions subsequent computation, and addressable prefix memory from which future states can retrieve context. We therefore factorize the sender state as

$$\mathcal{S}_i = (R_i, C_i), \tag{4}$$

where R_i is the *continuation seed* and C_i is the *context bank*.

Continuation seed. Let F denote the final prefill position after agent A_i processes its local input x_i together with the incoming latent message $\mathcal{M}_{i-1}$. We form the continuation seed by stacking the hidden states at this boundary across all L backbone layers:

$$R_i = \left[h_{i,F}^1; h_{i,F}^2; \ldots; h_{i,F}^L\right] \in \mathbb{R}^{L \times d}, \tag{5}$$

where d is the hidden dimension. Unlike a final-layer-only readout, R_i exposes the writer to the full layer-wise representation at the continuation boundary, providing a compact summary from which the outgoing continuation can be constructed.

Context bank. The boundary representation alone does not provide the addressable prefix memory used by Transformer attention. We therefore retain a compressed KV context bank,

$$C_i = \left\{ \left(K_{i,n}^{\ell}, V_{i,n}^{\ell} \right) \right\}_{\substack{\ell=1,\dots,L \\ n=1,\dots,N_i^C}}, \tag{6}$$

where N_i^C is the number of retained context positions. Thus, R_i summarizes the computation at the continuation boundary, while C_i preserves contextual information in an explicitly retrievable form. This factorization allows the writer to condition continuation construction on R_i without requiring the seed itself to encode the entire prefix. In our implementation, C_i is instantiated using Activation Beacon (Zhang et al., 2024). Details of beacon training, compression, and target extraction are provided in Appendix B.1.

3.3 One-Shot Thought Block Writer

Autoregressive continuation uses R_i and C_i through a sequential process. Starting from the continuation boundary, each newly generated latent state becomes part of the computation used to construct subsequent states, while the prefix memory remains accessible throughout the rollout. The resulting trajectory therefore performs two functions jointly: it *expands* the current computational state into future positions and *contextualizes* those positions using the available prefix memory.

Once the autoregressive rollout is removed, these two functions must be implemented differently. Thought Blocks separate them into *parallel continuation expansion* and *memory-backed rendering*.

Parallel continuation expansion. Autoregressive decoding obtains multiple future positions through temporal recurrence. Without this recurrence, a single continuation seed must instead be expanded directly into multiple latent positions. We introduce B learnable slots

$$O = [o_1, o_2, \dots, o_B] \in \mathbb{R}^{B \times d_s}, \tag{7}$$

where each slot indexes one position in the outgoing Thought Block. The slots act as position-specific queries over the layer-wise continuation seed R_i. We construct B sender-conditioned block embeddings in parallel through a lightweight cross-attention module,

$$E_i = g_\phi(O, R_i) \in \mathbb{R}^{B \times d}, \tag{8}$$

where O provides the queries and R_i provides the keys and values.

Equation 8 replaces *temporal state expansion* with *parallel positional expansion*. Rather than producing future positions through recurrent state transitions, the B slots query the same continuation boundary simultaneously. Because R_i contains boundary representations from all backbone layers, different slots can extract different combinations of layer-wise information, yielding distinct sender-conditioned block embeddings without recurrence across block positions.

Importantly, this expansion is intentionally driven by R_i rather than requiring E_i to encode the entire preceding context. The role of E_i is to specify *where the computation should continue*, not to reproduce everything the model may subsequently need to remember.

Memory-backed rendering. The expanded block embeddings E_i specify B candidate continuation positions, but they do not yet constitute model-native continuation memory. We therefore render them through the frozen backbone while retaining access to the context bank C_i. Specifically, we treat E_i as the input embeddings of B virtual continuation positions and perform a single causal Transformer forward pass. At each layer, the retained keys and values in C_i are supplied as prefix KV memory, allowing the block positions to retrieve information from the compressed prefix through the backbone's native self-attention. The complete writer is therefore

$$\mathcal{M}_i = \mathcal{W}_{\phi;\theta^\star}(R_i, C_i) = \mathrm{Render}_{\theta^\star}(E_i; C_i), \qquad E_i = g_\phi(O, R_i), \tag{9}$$

where $\mathrm{Render}_{\theta^\star}$ denotes this single frozen-backbone forward pass. During rendering, the Transformer materializes a key and value at every block position and every layer. We collect these states as the outgoing latent message,

$$\mathcal{M}_i = \left\{ \left(\widehat{K}_{i,j}^{\ell}, \widehat{V}_{i,j}^{\ell} \right) \right\}_{\substack{\ell=1,\dots,L \\ j=1,\dots,B}}. \tag{10}$$

Although the B continuation positions interact under the backbone's standard causal attention mask, they are processed within a single forward pass rather than generated through B autoregressive decode steps.

3.4 TRAJECTORY-TO-STATE DISTILLATION

Equation 9 specifies how to construct continuation memory directly. We now describe how the writer learns which continuation state to construct.

Teacher state construction. Starting from the same factorized continuation state (R_i, C_i), the frozen backbone performs autoregressive latent decoding following the continuous latent-feedback interface used in autoregressive latent communication (Zou et al., 2025):

$$\tau_i = [u_{i,1}, \ldots, u_{i,T_{\mathrm{lat}}}] = \mathrm{ARDecode}_{\theta^\star}(R_i; C_i). \tag{11}$$

The trajectory contains a sequence of intermediate states, whereas the student communicates a fixed-size block. We therefore apply a state-extraction operator Π to obtain a fixed-size teacher target:

$$\mathcal{M}_i^\star = \Pi(\tau_i). \tag{12}$$

Importantly, the student is not trained to reproduce the trajectory τ_i itself. The trajectory serves only as the computational path through which the teacher reaches $\mathcal{M}_i^\star$. The target of distillation is the resulting continuation state.

Direct continuation-state reconstruction. The student receives only the pre-trajectory state (R_i, C_i) and directly constructs $\widehat{\mathcal{M}}_i = \mathcal{W}_{\phi;\theta^\star}(R_i, C_i)$. Let $\widehat{K}_{i,j}^{\ell}, \widehat{V}_{i,j}^{\ell}$ denote the student KV states at block position j and layer ℓ, and let $K_{i,j}^{\ell,\star}, V_{i,j}^{\ell,\star}$ denote the corresponding teacher targets. The primary distillation objective is

$$\mathcal{L}_{\mathrm{KV}} = \frac{1}{BL} \sum_{\ell=1}^{L} \sum_{j=1}^{B} \left(\left\| \widehat{K}_{i,j}^{\ell} - K_{i,j}^{\ell,\star} \right\|_F^2 + \left\| \widehat{V}_{i,j}^{\ell} - V_{i,j}^{\ell,\star} \right\|_F^2 \right). \tag{13}$$

This objective directly matches the representation consumed by the downstream agent. The teacher reaches this target only after T_{lat} sequential latent steps, whereas the student learns to construct it directly from the continuation boundary state. We additionally use an auxiliary hidden-state alignment objective during training. Because this auxiliary state is neither communicated nor required at inference, its definition is deferred to Appendix B.1. The full objective is

$$\mathcal{L}_{\mathrm{TSD}} = \lambda_{\mathrm{KV}} \mathcal{L}_{\mathrm{KV}} + \lambda_{\mathrm{aux}} \mathcal{L}_{\mathrm{aux}}. \tag{14}$$

Only the writer parameters ϕ are optimized during this stage; the language-model backbone and contextual-memory mechanism remain fixed. Trajectory-to-state distillation therefore induces an asymmetric teacher–student computation: the teacher executes a sequential trajectory to reach a useful continuation state, while the student learns the direct mapping $(R_i, C_i) \longrightarrow \widehat{\mathcal{M}}_i$. At inference time, the teacher trajectory and auxiliary training signal are removed entirely.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Tasks and datasets. We evaluate on eight datasets spanning three task categories: (1) **Science, Commonsense, and Mathematical Reasoning**, including ARC (Clark et al., 2018), OpenBookQA (Mihaylov et al., 2018), and GSM8K (Cobbe et al., 2021); (2) **Knowledge-Intensive Question Answering**, including MMLU (Hendrycks et al., 2021), GPQA (Rein et al., 2024), and MedQA (Jin et al., 2021); and (3) **Reading Comprehension**, including BoolQ (Clark et al., 2019) and QuAIL (Rogers et al., 2020). We evaluate ARC on both the Challenge and Easy splits and MedQA on its US, Taiwan, and Mainland variants, yielding 11 benchmark configurations in total.

Baselines. We compare against three baselines. (1) **Single-agent**: a single model using the same backbone and a matched total decoding budget. (2) **Text-based multi-agent (Text-MAS)**: agents follow the same role decomposition as our method but exchange natural-language messages. (3) **Autoregressive latent communication (Latent-MAS)**: intermediate agents communicate through autoregressively generated latent thoughts (Zou et al., 2025). All multi-agent methods use the same backbone, role decomposition, role prompts, and number of agents, so they differ primarily in their inter-agent communication mechanism.

Table 2: Main comparison across four LLM backbones and three task categories. Results are category means. *Improve* compares Ours with Text-MAS: accuracy is Ours − Text-MAS, efficiency metrics are relative reductions, and latency is speedup. Bold indicates the best multi-agent result.

		Qwen2.5-7B					Qwen3-8B				
Task	**Method**	**Acc.**	**Text Tokens**	**Passes***	**AR Steps†**	**Latency†**	**Acc.**	**Text Tokens**	**Passes***	**AR Steps†**	**Latency†**
Reasoning	Single Text	69.06	233	-	53	2.18	86.45	233	-	50	1.75
	Text-MAS	**74.91**	1717	483	492	16.09	**89.75**	1546	375	380	16.80
	Latent-MAS	73.85	751	12	30	3.85	89.62	752	12	16	3.99
	Ours	73.77	**737**	**3**	**6**	**0.85**	89.24	**738**	**3**	**10**	**1.23**
	Improve	-1.13	57.1%	99.4%	98.7%	18.88×	-0.51	52.3%	99.2%	97.3%	13.66×
Question Answering	Single Text	56.12	285	-	11	0.42	62.57	397	-	126	6.17
	Text-MAS	58.16	2298	595	618	20.42	70.78	2252	552	574	31.31
	Latent-MAS	**58.90**	1117	12	25	3.75	**70.94**	1156	12	**15**	3.37
	Ours	57.89	**1114**	**3**	**7**	**1.05**	70.27	**1136**	**3**	31	**3.20**
	Improve	-0.27	51.5%	99.5%	98.9%	19.37×	-0.50	49.5%	99.5%	94.6%	9.79×
Reading	Single Text	74.66	439	-	6	0.24	80.23	452	-	15	0.60
	Text-MAS	77.36	2502	356	370	12.95	**84.39**	2478	324	346	16.36
	Latent-MAS	**79.31**	**1753**	12	17	3.70	84.06	1815	12	14	3.95
	Ours	75.63	**1753**	**3**	**7**	**1.61**	83.84	**1749**	**3**	**6**	**1.08**
	Improve	-1.73	29.9%	99.2%	98.1%	8.02×	-0.55	29.4%	99.1%	98.3%	15.08×

		Qwen3-14B					Mistral-7B				
Task	**Method**	**Acc.**	**Text Tokens**	**Passes***	**AR Steps†**	**Latency†**	**Acc.**	**Text Tokens**	**Passes***	**AR Steps†**	**Latency†**
Reasoning	Single Text	85.19	222	-	34	1.79	59.25	270	-	63	2.17
	Text-MAS	89.44	1765	491	496	26.14	65.27	2994	704	764	40.89
	Latent-MAS	**91.40**	**771**	12	155	17.01	**66.44**	**1405**	12	581	29.94
	Ours	89.99	850	**3**	**107**	**9.33**	65.72	1407	**3**	**286**	**13.12**
	Improve	+0.55	51.8%	99.4%	78.4%	2.80×	+0.45	53.0%	99.6%	62.6%	3.12×
Question Answering	Single Text	69.46	318	-	44	2.07	39.68	447	-	117	4.99
	Text-MAS	**74.64**	2565	692	711	38.34	44.89	4300	993	1068	62.81
	Latent-MAS	73.91	**1158**	12	115	16.01	**46.85**	1831	12	585	38.66
	Ours	74.40	1189	**3**	**98**	**8.40**	46.28	**1806**	**3**	**275**	**16.46**
	Improve	-0.24	53.7%	99.6%	86.1%	4.56×	+1.40	58.0%	99.7%	74.3%	3.82×
Reading	Single Text	82.78	448	-	7	0.44	71.28	512	-	32	1.03
	Text-MAS	**86.03**	2621	412	418	22.81	73.86	3666	588	631	34.45
	Latent-MAS	85.72	**1810**	12	141	13.85	**75.34**	2487	12	581	35.66
	Ours	85.95	1852	**3**	**111**	**11.02**	74.69	**2475**	**3**	**272**	**15.39**
	Improve	-0.08	29.3%	99.3%	73.5%	2.07×	+0.83	32.5%	99.5%	56.9%	2.24×

Note. * Communication-stage metric, measured over intermediate-agent message construction. † End-to-end metric, measured from input to final output; latency is in seconds.

Metrics. We evaluate task utility and inference efficiency, distinguishing *end-to-end* from *communication-stage* cost. Task performance is measured by **accuracy**, using the official evaluation protocol where available and otherwise deterministic exact matching against official gold labels. For end-to-end efficiency, we report **text tokens**, **AR Steps**, and **latency**. Text tokens count tokenizer-visible tokens generated by all agents, including input text tokens and output text tokens; AR Steps count autoregressive decoding steps across intermediate communication and final-answer generation; and latency measures wall-clock time for the full pipeline, including prefill, communication, and final decoding.

To capture sequential communication cost not reflected by token counts, we also report **communication forward passes (Passes)**, defined as the number of intermediate-agent forward passes used to construct outgoing messages. Text-MAS and Latent-MAS require sequential passes, whereas Thought Blocks use one non-autoregressive rendering pass per intermediate agent and therefore incur zero autoregressive communication steps. We additionally report **KV-cache load**, the amount of KV state loaded during decoding. All latency measurements use the same hardware and configuration and implementation details are provided in Appendix B.5.

4.2 MAIN RESULTS

Table 2 compares Thought Blocks with text-based and autoregressive latent multi-agent communication across four LLM backbones[1]. Overall, Thought Blocks retain comparable task performance while substantially reducing sequential communication and end-to-end decoding cost. Relative to Latent-MAS, the more direct comparison for isolating message construction in latent space, Thought Blocks reduce communication passes from 12 to 3 under our four-agent setup (75.0%), reduce end-to-end autoregressive steps by 35.7% on average, and achieve a $2.5\times$ average latency speedup, with an average accuracy difference of -0.72 points. Relative to Text-MAS, Thought Blocks reduce text tokens by 45.7%, communication passes by 99.4%, and end-to-end autoregressive steps by 84.8%, while achieving an $8.6\times$ average latency speedup with an average accuracy difference of only -0.15 points. Compared with the decoding-budget-matched Single Text baseline, Thought Blocks achieve higher accuracy in all 12 backbone–category settings, with a $+4.25$-point average improvement.

These comparisons separate two sources of communication overhead. Latent communication removes explicit natural-language verbalization, but Latent-MAS still constructs each inter-agent message through a sequential latent rollout. Thought Blocks target this remaining construction cost by replacing the rollout with a single non-autoregressive block-construction pass per inner agent. Under our four-agent setup, this reduces communication-stage passes from 12 for Latent-MAS to 3 for Thought Blocks; Text-MAS requires hundreds of such passes because its intermediate messages are decoded autoregressively. Notably, Thought Blocks and Latent-MAS have the same macro-average text-token count in Table 2, yet Thought Blocks remain substantially faster. This comparison therefore shows that the efficiency gain is not explained by shorter textual outputs, but by replacing sequential latent message construction with one-shot block construction.

4.3 ABLATION STUDY

We ablate two design choices of Thought Blocks on Qwen3-8B with $B = 4$. First, we test whether the block positions require writer-generated content by replacing the block embeddings E_i with zero or random vectors while leaving the rest of the pipeline unchanged. Second, we evaluate the factorized continuation state $\mathcal{S}_i = (R_i, C_i)$ by retaining only R_i or only C_i.

As shown in Table 3, replacing the writer-generated content of E_i consistently degrades performance across all eleven benchmark configurations. The average accuracy drops from 79.64 for the full method to 73.44 with ZERO-E (-6.20 points) and to 71.06 with RANDOM-E (-8.58 points). Moreover, RANDOM-E underperforms ZERO-E on all eleven benchmarks, with an additional average drop of 2.38 points. These results rule out the possibility that the gains of Thought Blocks arise merely from introducing additional latent positions. Instead, the content constructed at these positions is functionally important, and replacing it with arbitrary latent values can interfere with downstream computation.

Table 3: Ablation study of our method and its variants on the datasets. Accuracy (%) is reported.

Method	ARC-C	ARC-E	BoolQ	GPQA	GSM8K	MedQA(US)	MedQA(TW)	MedQA(ML)	MMLU	OpenBookQA	QuAIL
Ours	91.76	96.91	90.25	43.89	82.05	71.16	73.41	80.12	82.79	86.23	77.44
Zero E	83.00	84.96	83.89	32.54	70.40	68.01	71.72	79.22	80.09	82.71	71.25
Random E	80.70	83.25	79.52	30.37	70.56	66.35	68.81	76.43	77.61	78.74	69.28
$R-only$	79.11	78.63	71.39	15.31	58.94	54.66	57.98	68.25	65.72	64.43	59.67
$C-only$	82.63	82.94	76.08	19.15	66.36	60.32	64.57	75.61	72.43	73.65	64.98

The state-factorization ablations show that both components are important for recovering the performance of the full method. Retaining only R_i reduces the average accuracy to 61.28 (-18.36 points), while retaining only C_i yields 67.16 (-12.48 points). The C-only variant outperforms the R-only variant on all eleven benchmarks, by 5.88 points on average, suggesting that the addressable contextual memory carried by C_i is particularly valuable when each component is considered in isolation. Nevertheless, both single-component variants remain substantially below the full method. This is consistent with the intended factorization: R_i provides the continuation signal used to construct the block positions, whereas C_i preserves addressable prefix memory that those positions can retrieve during rendering. Full performance therefore depends on using the two components jointly.

[1]Full results across all benchmarks are provided in Appendix Table B.1.

4.4 Effect of Thought Block Size

We investigate how the Thought Block size B affects the trade-off between communication capacity and efficiency on Qwen3-8B. As shown in Table 4, even $B = 1$ retains substantial task utility, achieving 78.05% dataset-macro average accuracy. Increasing B to 4 improves accuracy to 81.12%, while further increasing it to 32 yields only modest additional gains, reaching 82.59%.

Meanwhile, latency remains nearly constant because all block positions are rendered in a single backbone forward pass, whereas KV-cache load grows with B. The strong $B = 1$ result is also consistent with our design: each slot is rendered into a full layer-wise KV continuation position conditioned on the compressed context bank. Overall, $B = 4$ provides a favorable balance between accuracy and memory cost.

Table 4: Effect of Thought Block size on Qwen3-8B.

Task Category	Metric	B=1	B=4	B=8	B=12	B=16	B=24	B=32
Reasoning	Accuracy	87.14	89.24	89.46	89.69	89.35	89.34	**90.02**
	Latency (s)	**1.13**	1.23	1.26	1.27	1.21	1.25	1.29
	KV Load (MB)	4.66	5.71	7.12	8.52	9.93	12.74	15.55
Question Answering	Accuracy	65.49	70.27	71.24	71.23	71.66	71.82	**72.21**
	Latency (s)	**3.12**	3.20	3.23	3.27	3.23	3.20	3.34
	KV Load (MB)	7.37	8.42	9.83	11.23	12.64	15.45	18.26
Reading	Accuracy	81.53	83.84	84.58	84.41	84.72	85.04	**85.55**
	Latency (s)	**0.94**	1.08	1.12	1.10	1.10	1.13	1.14
	KV Load (MB)	12.39	13.44	14.84	16.24	17.66	20.47	23.28

These results show diminishing returns from increasing the Thought Block size: even $B = 4$ achieves most of the accuracy obtained with larger blocks, while KV-cache load increases approximately linearly with B. Thus, within the tested range, B mainly controls the accuracy–memory trade-off rather than acting as a highly performance-sensitive hyperparameter.

4.5 Decoding-Time KV-Cache Load

Table 5 reports decoding-time KV-cache load. Thought Blocks load 8.35 MB on average, compared with 58.29 MB for Latent-MAS and 71.13 MB for Text-MAS, reducing loaded KV-cache volume by 85.7% and 88.3%, respectively. This shows that the efficiency gain extends beyond token count and sequential forward steps to substantially lower decoding-time KV-cache load.

Table 5: Loaded KV-cache size (MB) during decoding on Qwen3-8B.

Method	ARC-C	ARC-E	BoolQ	GPQA	GSM8K	MedQA(ML)	MedQA(TW)	MedQA(US)	MMLU	OpenBookQA	QuAIL
Ours	**6.25**	**5.66**	**8.61**	**10.13**	**5.61**	**6.78**	**7.28**	**10.76**	**7.17**	**5.33**	**18.26**
Latent-MAS	41.96	39.95	59.48	71.42	38.99	47.08	51.14	75.68	52.87	36.42	126.18
Text-MAS	58.02	55.49	64.83	93.86	50.43	67.54	75.49	89.27	65.75	52.79	108.91

5 Discussion and Limitations

Our results support a distinction between message representation and message construction. Thought Blocks show that useful continuation states can be constructed directly from the sender state without executing an autoregressive latent rollout at inference, while preserving comparable downstream utility across multiple backbones and tasks. This suggests that autoregression is one mechanism for reaching a useful latent state, rather than an intrinsic requirement of latent communication. The results nevertheless reflect an accuracy–efficiency trade-off: performance varies across tasks and models, and larger blocks provide diminishing accuracy gains at increasing KV-memory cost.

Our study focuses on establishing whether autoregressive latent message construction can be compiled into a direct continuation-state mapping. The current training pipeline uses autoregressive teacher rollouts to provide offline state-level supervision, but this sequential computation is removed entirely at inference. Consequently, the additional training cost is a one-time overhead whose importance depends on deployment scale; our amortization analysis in Appendix B.4 suggests that it becomes relatively small in sufficiently high-volume multi-agent workloads. We evaluate this principle in controlled sequential, shared-backbone systems in order to isolate message-construction effects.

AI USE STATEMENT

In this work, we used generative AI tools for tasks requiring disclosure, including refining research hypotheses, providing feedback on experimental and ablation design, and assisting with the interpretation and consistency checking of experimental results. We did not use generative AI tools to generate synthetic datasets, formulate mathematical claims, provide critical ingredients for or write mathematical proofs, or perform qualitative or thematic data analysis. Other required-disclosure tasks, including those involving surveys or interviews, transcription of research material, and human-subject data, are not applicable to this work.

Additionally, we used generative AI tools for recommended-disclosure tasks, including drafting and editing portions of the manuscript, improving readability and organization, language polishing and technical rephrasing, checking notation and numerical consistency, formatting LaTeX tables and appendix material, and assisting with the design and refinement of scientific figures. All AI-assisted outputs were reviewed and verified by the authors. Numerical values and reported comparisons were checked against the underlying experimental outputs; technical descriptions and notation were checked against the method and implementation; and manuscript edits, tables, and figures were manually inspected for correctness, consistency, and fidelity to the reported experiments. The authors take full responsibility for the final content of this work, including any text or artifacts produced with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

We have organized the paper and supplementary material to support reproducibility of both the proposed method and the reported experiments. The complete formulation of Thought Blocks, including the factorized continuation state, one-shot Thought Block writer, and trajectory-to-state distillation objective, is provided in Sec. 3. The experimental setup in Sec. 4.1 specifies the evaluated datasets, baseline construction, shared multi-agent configuration, evaluation metrics, and efficiency measurements. Appendix B.1 provides the full training procedure, including Beacon training, autoregressive teacher-state construction, state extraction, and the objectives used to train the Thought Block writer. Appendix B.2 reports the agent prompts used in our experiments, while Appendix B.5 documents the hardware environment and key implementation and optimization settings. The appendix further provides full per-dataset results and additional ablations underlying the aggregated results reported in the main text. Together, these sections provide the information needed to reconstruct the proposed communication mechanism, reproduce its training and evaluation protocol, and verify the reported effectiveness and efficiency results. After the paper is accepted, the code project will be made open source.

REFERENCES

Siyi Chen, Xiaoyan Zhang, Meng Wu, Jonathan Tremblay, Valts Blukis, Stan Birchfield, Rene Vidal, Alvaro Velasquez, Sijia Liu, and Qing Qu. See what i see, know what i think: Dense latent communication across heterogeneous agents. *arXiv preprint arXiv:2606.13594*, 2026. URL `https://arxiv.org/abs/2606.13594`.

Weize Chen, Jiarui Yuan, Chen Qian, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Optima: Optimizing Effectiveness and Efficiency for LLM-Based Multi-Agent System. *arXiv preprint arXiv:2410.08115*, 2024.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2924–2936, 2019.

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

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.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

Zhuoyun Du, Runze Wang, Huiyu Bai, Zouying Cao, Xiaoyong Zhu, Yu Cheng, Bo Zheng, Wei Chen, and Haochao Ying. Enabling agents to communicate entirely in latent space. *arXiv preprint arXiv:2511.09149*, 2025.

Tianyu Fu, Zihan Min, Hanling Zhang, Jichao Yan, Guohao Dai, Wanli Ouyang, and Yu Wang. Cache-to-Cache: Direct semantic communication between large language models. *arXiv preprint arXiv:2510.03215*, 2025. URL https://arxiv.org/abs/2510.03215. Published at ICLR 2026.

Halil Alperen Gozeten, Muhammed Emrullah Ildiz, Xuechen Zhang, Hrayr Harutyunyan, Ankit Singh Rawat, and Samet Oymak. Continuous chain of thought enables parallel exploration and reasoning. In *The Fourteenth International Conference on Learning Representations*, 2026.

Wes Gurnee, Nicholas Sofroniew, Adam Pearce, Mateusz Piotrowski, Isaac Kauvar, Runjin Chen, Anna Soligo, Paul Bogdan, Euan Ong, Rowan Wang, Ben Thompson, David Abrahams, Subhash Kantamneni, Emmanuel Ameisen, Joshua Batson, and Jack Lindsey. Verbalizable representations form a global workspace in language models. *Transformer Circuits Thread*, 2026. URL https://transformer-circuits.pub/2026/workspace/index.html.

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.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021.

Haoqiang Kang, Yizhe Zhang, Nikki Lijing Kuang, Nicklas Majamaki, Navdeep Jaitly, Yi-An Ma, and Lianhui Qin. LaDiR: Latent diffusion enhances LLMs for text reasoning. *arXiv preprint arXiv:2510.04573*, 2025.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems*, 2023.

Yiping Li, Zhiyu An, and Wan Du. When less latent leads to better relay: Information-preserving compression for latent multi-agent LLM collaboration. *arXiv preprint arXiv:2604.13349*, 2026.

Jiyao Liu, Qi Zhang, Yaoyi Jia, Ziwen Kan, and Song Wang. Dual-cache latent space communication between heterogeneous language models. *arXiv preprint arXiv:2608.20617*, 2026.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic LLM-powered agent network for task-oriented agent collaboration. *arXiv preprint arXiv:2310.02170*, 2023.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, 2018.

OpenAI. On the Navier–Stokes millennium prize problem, September 2026. OpenAI Research, published September 8, 2026.

Yanwen Peng, Delvin Ce Zhang, Xi Wang, and Nikolaos Aletras. StateBridge: Training-free hidden-state alignment for latent communication in LLM multi-agent systems. In *Conference on Language Modeling*, 2026.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative Agents for Software Development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15174–15186, 2024.

Vignav Ramesh and Kenneth Li. Communicating activations between language model agents. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 51094–51116. PMLR, 2025.

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. In *First Conference on Language Modeling*, 2024.

Anna Rogers, Olga Kovaleva, Matthew Downey, and Anna Rumshisky. Getting closer to ai complete question answering: A set of prerequisite real tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8722–8731, 2020.

Maximillian Rossi, Prajwal Raghunath, and Eugene Wu. Latent Cache Flow: Model-to-model communication without text. *arXiv preprint arXiv:2605.22863*, 2026. URL `https://arxiv.org/abs/2605.22863`.

Xiangyu Shi, Marco Chiesa, Gerald Q. Maguire Jr., and Dejan Kostic. KVComm: Enabling efficient LLM communication through selective KV sharing. In *The Fourteenth International Conference on Learning Representations*, 2026.

Zhexuan Wang, Yutong Wang, Xuebo Liu, Liang Ding, Miao Zhang, Jie Liu, and Min Zhang. AgentDropout: Dynamic Agent Elimination for Token-Efficient and High-Performance LLM-Based Multi-Agent Collaboration. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 24013–24035, 2025.

Haoyi Wu, Zhihao Teng, and Kewei Tu. Parallel continuous chain-of-thought with jacobi iteration. *arXiv preprint arXiv:2506.18582*, 2025.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Bingyu Yan, Xiaoming Zhang, Litian Zhang, Lian Zhang, Ziyi Zhou, Dezhuang Miao, and Chaozhuo Li. Beyond self-talk: A communication-centric survey of LLM-based multi-agent systems. *arXiv preprint arXiv:2502.14321*, 2025.

Hancheng Ye, Zhengqi Gao, Mingyuan Ma, Qinsi Wang, Yuzhe Fu, Ming-Yu Chung, Yueqian Lin, Zhijian Liu, Jianyi Zhang, Danyang Zhuo, and Yiran Chen. KVCOMM: Online Cross-Context KV-Cache Communication for Efficient LLM-Based Multi-Agent Systems. *arXiv preprint arXiv:2510.12872*, 2025.

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

Yujia Zheng, Zhuokai Zhao, Zijian Li, Yaqi Xie, Mingze Gao, Lizhu Zhang, and Kun Zhang. Thought communication in multiagent collaboration. *arXiv preprint arXiv:2510.20733*, 2025.

Jiaru Zou, Xiyuan Yang, Ruizhong Qiu, Gaotang Li, Katherine Tieu, Pan Lu, Ke Shen, Hanghang Tong, Yejin Choi, James Zou, Mengdi Wang, and Ling Yang. Latent collaboration in multi-agent systems. *arXiv preprint arXiv:2511.20639*, 2025.

APPENDIX

A METHOD OVERVIEW

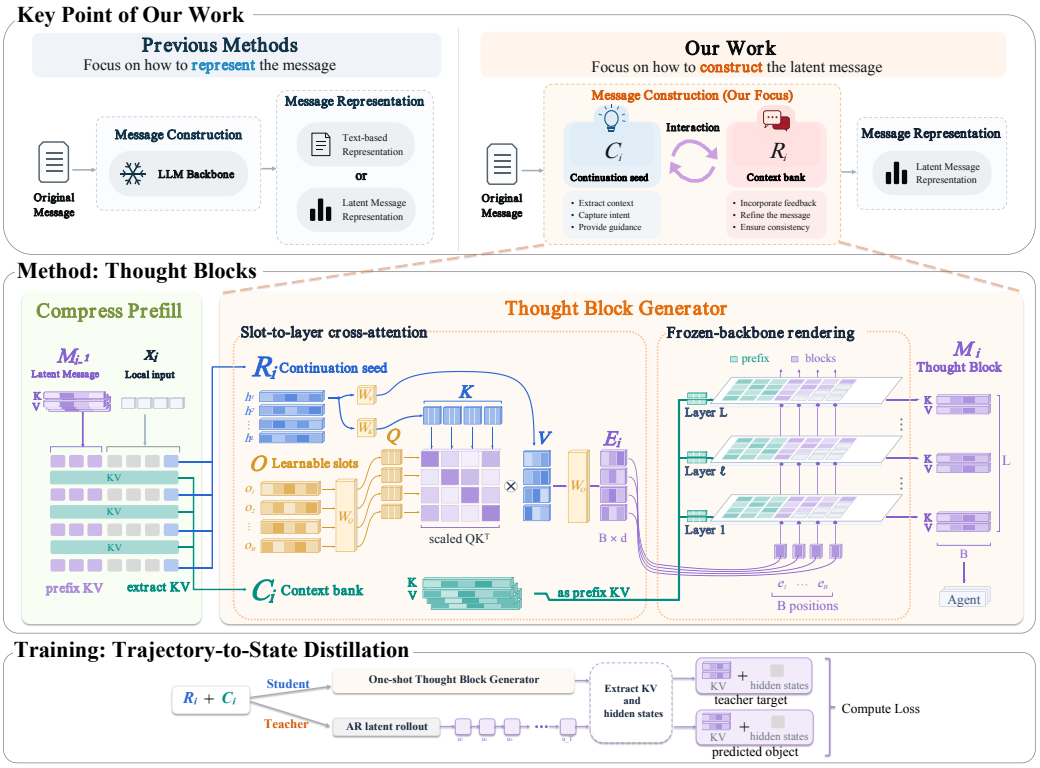


Figure A.1:

B EXPERIMENT SETUPS

B.1 TRAINING THOUGHT BLOCKS

Training follows the same trajectory-to-state view introduced in Sec. 3.4. Stage I trains the Activation Beacon mechanism used to instantiate the compressed context bank while keeping the language-model backbone fixed. Stage II then performs trajectory-to-state distillation. Starting from the factorized continuation state $\mathcal{S}_i = (R_i, C_i)$, a frozen autoregressive latent teacher executes the trajectory τ_i and a state-extraction operator Π converts that trajectory into a fixed-size teacher continuation-memory target $\mathcal{M}_i^\star$. The one-shot Thought Block writer receives only (R_i, C_i) and is trained to reconstruct this target directly. The teacher trajectory is therefore used only to construct supervision during training and is removed entirely at inference time.

B.1.1 STAGE I: BEACON TRAINING

We instantiate the context bank C_i with Activation Beacon (Zhang et al., 2024). Let θ denote the backbone parameters and let β denote the trainable Beacon mechanism, including the Beacon embedding and Beacon-specific projections. Given a training corpus $\mathcal{D}$, we optimize only β with the Activation Beacon objective,

$$\beta^\star = \arg\min_{\beta} \mathbb{E}_{x\sim\mathcal{D}} \left[\mathcal{L}_{\mathrm{AB}}\big(\mathrm{Beacon}_\beta(x); \theta\big) \right]. \tag{15}$$

The language-model weights are not updated in this stage. We use $\theta^\star$ to denote this frozen backbone in the remainder of the paper and freeze the trained Beacon mechanism $\beta^\star$ as well. Consequently, Stage II optimizes only the writer parameters ϕ.

B.1.2 Stage II-A: Teacher State Construction

For each training example x_i, we first run the same Beacon-compressed prefill used by the main method to obtain the pre-trajectory continuation state

$$\mathcal{S}_i = (R_i, C_i), \tag{16}$$

where R_i is the layer-wise continuation seed at the final prefill position and C_i is the compressed, addressable prefix KV memory defined in Sec. 3.2.

The frozen backbone then performs autoregressive latent decoding from this state. Following autoregressive latent communication (Zou et al., 2025), the last-layer teacher hidden state is mapped back to the model input-embedding space before being used as the next continuous input. Under the row-vector convention, we use the fixed alignment operator

$$W_a = W_{\mathrm{out}}^{\dagger} W_{\mathrm{in}} \in \mathbb{R}^{d \times d}, \tag{17}$$

where $W_{\mathrm{in}}, W_{\mathrm{out}} \in \mathbb{R}^{|\mathcal{V}| \times d}$ are the backbone input- and output-embedding matrices and $W_{\mathrm{out}}^{\dagger}$ is the Moore–Penrose pseudoinverse. If $h_{i,t}^{\mathrm{teach}}$ is the last-layer teacher hidden state at latent step t, the continuous latent state used by the next step is

$$u_{i,t} = h_{i,t}^{\mathrm{teach}} W_a. \tag{18}$$

The resulting teacher trajectory is exactly the trajectory used in the main text,

$$\tau_i = [u_{i,1}, \ldots, u_{i,T_{\mathrm{lat}}}] = \mathrm{ARDecode}_{\theta^\star}(R_i; C_i). \tag{19}$$

To instantiate the state-extraction operator Π in Eq. 12, we insert decode-stage Beacons along the teacher rollout and retain the states at the first B valid Beacon positions. The extracted teacher continuation-memory target is

$$\mathcal{M}_i^\star = \Pi(\tau_i) = \left\{ \left(K_{i,j}^{\ell,\star}, V_{i,j}^{\ell,\star} \right) \right\}_{\substack{\ell=1,\ldots,L \\ j=1,\ldots,B}}. \tag{20}$$

At the same retained positions, we additionally record the corresponding last-layer hidden states

$$Z_i^\star = \left\{ z_{i,j}^\star \right\}_{j=1}^{B}, \tag{21}$$

which are used only for the auxiliary training loss and are not part of the communicated message. Thus, each collected supervision record contains $(R_i, C_i, \mathcal{M}_i^\star, Z_i^\star)$. This construction makes the distillation target model-native: the writer learns to reconstruct continuation states actually reached by the frozen autoregressive teacher rather than arbitrary latent tensors.

B.1.3 Stage II-B: Thought Block Writer Training

Given (R_i, C_i), the writer first performs the same parallel continuation expansion as in Sec. 3.3,

$$E_i = g_\phi(O, R_i), \tag{22}$$

and then renders the B virtual continuation positions through the frozen backbone conditioned on C_i,

$$\widehat{\mathcal{M}}_i = \mathrm{Render}_{\theta^\star}(E_i; C_i). \tag{23}$$

The same forward pass also produces the last-layer hidden states used by the auxiliary objective. We denote them by

$$\widehat{Z}_i = \{\hat{z}_{i,j}\}_{j=1}^{B}, \qquad \widehat{\mathcal{M}}_i = \left\{ \left(\widehat{K}_{i,j}^{\ell}, \widehat{V}_{i,j}^{\ell} \right) \right\}_{\substack{\ell=1,\ldots,L \\ j=1,\ldots,B}}. \tag{24}$$

The primary objective is the same KV-state reconstruction loss defined in Eq. 13,

$$\mathcal{L}_{\mathrm{KV}} = \frac{1}{BL} \sum_{\ell=1}^{L} \sum_{j=1}^{B} \left(\left\| \widehat{K}_{i,j}^{\ell} - K_{i,j}^{\ell,\star} \right\|_F^2 + \left\| \widehat{V}_{i,j}^{\ell} - V_{i,j}^{\ell,\star} \right\|_F^2 \right). \tag{25}$$

The auxiliary hidden-state alignment objective deferred from the main text is

$$\mathcal{L}_{\text{aux}} = \frac{1}{B} \sum_{j=1}^{B} \left\| \hat{z}_{i,j} - z^{\star}_{i,j} \right\|_2^2 . \tag{26}$$

The complete trajectory-to-state distillation objective is therefore

$$\mathcal{L}_{\text{TSD}} = \lambda_{\text{KV}} \mathcal{L}_{\text{KV}} + \lambda_{\text{aux}} \mathcal{L}_{\text{aux}}, \tag{27}$$

which matches Eq. 14. Only ϕ is updated in Stage II; $\theta^{\star}$ and $\beta^{\star}$ remain frozen. At inference time, neither the autoregressive teacher trajectory τ_i nor the auxiliary target $Z_i^{\star}$ is needed: the writer directly applies $(R_i, C_i) \mapsto \mathcal{M}_i$ in one forward pass.

B.2 Prompt Template

Planner Agent Prompt

```
You are a Planner Agent.  Given an input question, design a clear,
step-by-step plan for how to solve the question.
Your outlined plan should be concise with a few bulletpoints for each
step.  Do not produce the final answer.
{question}
Now output your plan to solve the question below:
```

Critic Agent Prompt

```
You are a Critic Agent to evaluate the correctness of the input plan
for the given question and provide helpful feedback for improving the
plan.
The plan information is provided in latent KV representation format.
Review the plan and question and output:
(1) original plan contents
(2) constructive feedback on the original plan.
{question}
Now, output your response below:
```

Refiner Agent Prompt

```
You are a Refiner Agent to provide a refined step-by-step plan for
solving the given question.
You are provided with:
(1) latent-format information:  a previous plan with feedback
(2) text-format information:  the input question you need to solve.
{question}
Based on the input, write a refined and improved plan to solve the
question.  Make sure your output plan is correct and concise.
Now, output your refined plan below:
```

Judger Agent Prompt

```
You are a helpful assistant.  You are provided with latent information
for reference and a target question to solve.
The latent information might contain irrelevant contents.  Ignore it
if it is not helpful for solving the target question.
You must reason step-by-step to solve the provided Target Question
without outputting other irrelevant information.
{question}
Now, output your final answer and just output one answer:
```

B.3 Full Experiment Results

Table B.1: Main performance comparison on all models across benchmarks. For accuracy, Improve is computed as Ours − Text-MAS. For efficiency metrics, Improve reports the reduction or speedup of Ours relative to Text-MAS. The best and second-best results among Ours, Latent-MAS, and Text-MAS are highlighted in bold and underlined, respectively. For Improve rows with an accuracy difference greater than -1.5 points. (Part 1/6: ARC-C, ARC-E.)

Dataset	Model	Method	Acc.	Text Tokens	Comm. Passes	AR Steps	Latency
ARC-C	Qwen2.5-7B	Ours	82.38	**784**	3	**4**	**0.68**
		Latent-MAS	83.39	788	12	17	3.39
		Text-MAS	**83.96**	1757	479	482	13.36
		Single	81.19	199	-	5	0.09
		Improve	-1.58	55.4%	99.4%	99.6%	19.6×
	Qwen3-8B	Ours	91.76	**769**	3	**4**	**0.61**
		Latent-MAS	90.19	800	12	14	3.52
		Text-MAS	**92.06**	1653	417	419	15.65
		Single	89.10	210	-	10	0.36
		Improve	-0.30	53.5%	99.3%	99.5%	25.7×
	Qwen3-14B	Ours	90.14	879	3	**93**	**7.87**
		Latent-MAS	**91.24**	**821**	12	107	17.77
		Text-MAS	90.48	1951	553	557	29.44
		Single	89.13	214	-	11	0.60
		Improve	-0.34	54.9%	99.5%	83.3%	3.7×
	Mistral-7B	Ours	71.84	1456.78	3	292	17
		Latent-MAS	**72.27**	1480	12	585	38
		Text-MAS	71.76	**3278**	806	**862**	**52**
		Single	69.03	271	-	49	1
		Improve	+0.08	55.56	99.6%	66.13	3×
ARC-E	Qwen2.5-7B	Ours	91.07	**746**	3	**4**	**0.98**
		Latent-MAS	91.33	748	12	16	2.45
		Text-MAS	**92.21**	1644	444	449	15.40
		Single	91.81	187	-	7	0.13
		Improve	-1.14	54.6%	99.3%	99.6%	15.7×
	Qwen3-8B	Ours	96.91	**741**	3	**4**	**0.69**
		Latent-MAS	97.14	762	12	14	3.97
		Text-MAS	**97.35**	1583	399	404	17.11
		Single	94.17	203	-	18	0.62
		Improve	-0.44	53.2%	99.2%	99.5%	24.8×
	Qwen3-14B	Ours	**97.62**	824	3	**88**	**11.59**
		Latent-MAS	96.39	**774**	12	181	17.52
		Text-MAS	97.12	1855	529	532	28.05
		Single	93.28	198	-	7	0.39
		Improve	+0.50	55.6%	99.4%	83.5%	2.4×
	Mistral-7B	Ours	82.14	1446.18	3	288	10
		Latent-MAS	81.83	1414	12	580	19
		Text-MAS	**82.21**	**2992**	651	**751**	**23**
		Single	80.04	246	-	35	1
		Improve	-0.07	51.66	99.5%	61.65	2×

Table B.2: Main performance comparison on all models across benchmarks (continued; Part 2/6: BoolQ, GPQA).

Dataset	Model	Method	Acc.	Text Tokens	Comm. Passes	AR Steps	Latency
BoolQ	Qwen2.5-7B	Ours	84.19	**1118**	3	**5**	**1.49**
		Latent-MAS	**86.76**	1119	12	16	3.35
		Text-MAS	85.85	1938	400	404	13.92
		Single	83.73	280	-	6	0.16
		Improve	-1.66	42.3%	99.2%	99.3%	9.3×
	Qwen3-8B	Ours	90.25	**1121**	3	**6**	**0.68**
		Latent-MAS	**90.76**	1133	12	15	2.56
		Text-MAS	90.12	1849	337	342	20.36
		Single	87.49	292	-	13	0.52
		Improve	+0.13	39.4%	99.1%	98.8%	29.9×
	Qwen3-14B	Ours	91.33	1230	3	121	9.33
		Latent-MAS	91.97	**1153**	12	**116**	**8.64**
		Text-MAS	**92.14**	1964	394	398	21.25
		Single	88.29	288	-	6	0.29
		Improve	-0.81	37.4%	99.2%	69.6%	2.3×
	Mistral-7B	Ours	**87.96**	1829.51	3	248	17
		Latent-MAS	87.75	1837	12	583	**36**
		Text-MAS	86.58	**3017**	593	**642**	31
		Single	83.30	352	-	40	1
		Improve	+1.38	39.35	99.5%	61.37	2×
GPQA	Qwen2.5-7B	Ours	36.65	1354	3	**13**	**1.50**
		Latent-MAS	**37.25**	**1337**	12	34	4.46
		Text-MAS	35.59	2719	678	739	25.60
		Single	36.39	350	-	15	0.53
		Improve	+1.06	50.2%	99.6%	98.2%	17.1×
	Qwen3-8B	Ours	43.89	**1414**	3	106	7.19
		Latent-MAS	43.46	1461	12	**16**	**2.83**
		Text-MAS	**44.14**	2729	672	732	51.20
		Single	39.38	522	-	188	6.37
		Improve	-0.25	48.2%	99.6%	85.5%	7.1×
	Qwen3-14B	Ours	**53.67**	1466	3	**169**	**13.30**
		Latent-MAS	51.13	**1420**	12	196	21.91
		Text-MAS	52.86	3141	888	904	48.94
		Single	49.67	406	-	71	3.73
		Improve	+0.81	53.3%	99.7%	81.3%	3.7×
	Mistral-7B	Ours	38.84	2021.45	3	301	16
		Latent-MAS	**38.95**	2037	12	587	38
		Text-MAS	38.56	**4647**	1041	**1165**	**71**
		Single	36.61	597	-	224	7
		Improve	+0.28	56.50	99.7%	74.16	4×

Table B.3: Main performance comparison on all models across benchmarks (continued; Part 3/6: GSM8K, MedQA(ML)).

Dataset	Model	Method	Acc.	Text Tokens	Comm. Passes	AR Steps	Latency
GSM8K	Qwen2.5-7B	Ours	44.05	**740**	3	**14**	**0.91**
		Latent-MAS	**44.87**	787	12	73	6.34
		Text-MAS	43.06	1823	530	552	18.60
		Single	31.84	372	-	195	8.26
		Improve	+0.99	59.4%	99.4%	97.5%	20.4×
	Qwen3-8B	Ours	82.05	755	3	27	**2.41**
		Latent-MAS	**83.55**	**753**	12	**23**	4.28
		Text-MAS	83.45	1439	291	296	20.07
		Single	77.10	303	-	122	4.24
		Improve	-1.40	47.5%	99.0%	90.9%	8.3×
	Qwen3-14B	Ours	79.48	933	3	**148**	**9.99**
		Latent-MAS	**82.13**	**785**	12	224	24.35
		Text-MAS	78.54	1663	444	450	23.68
		Single	71.20	296	-	111	5.67
		Improve	+0.94	43.9%	99.3%	67.1%	2.4×
	Mistral-7B	Ours	36.47	1401.95	3	276	15
		Latent-MAS	**38.14**	1390	12	583	**37**
		Text-MAS	36.45	**2642**	578	**600**	37
		Single	25.09	329	-	124	5
		Improve	+0.02	46.94	99.5%	54.00	3×
MedQA(ML)	Qwen2.5-7B	Ours	82.96	**882**	3	**5**	**0.74**
		Latent-MAS	**85.65**	888	12	20	3.54
		Text-MAS	81.95	1996	558	569	15.86
		Single	80.30	224	-	8	0.23
		Improve	+1.01	55.8%	99.5%	99.5%	21.4×
	Qwen3-8B	Ours	80.12	901	3	**6**	**0.69**
		Latent-MAS	80.18	**897**	12	14	3.04
		Text-MAS	**81.41**	1930	514	522	26.03
		Single	77.05	294	-	74	3.70
		Improve	-1.29	53.3%	99.4%	99.2%	37.7×
	Qwen3-14B	Ours	83.03	952	3	**96**	**7.68**
		Latent-MAS	83.11	**888**	12	104	17.90
		Text-MAS	**84.68**	2321	685	693	36.88
		Single	78.17	233	-	10	0.54
		Improve	-1.65	59.0%	99.6%	86.1%	4.8×
	Mistral-7B	Ours	46.93	1673.19	3	257	15
		Latent-MAS	**47.85**	1682	12	585	38
		Text-MAS	41.41	**3893**	918	**975**	**61**
		Single	32.05	366	-	81	4
		Improve	+5.52	57.02	99.7%	73.64	4×

Table B.4: Main performance comparison on all models across benchmarks (continued; Part 4/6: MedQA(TW), MedQA(US)).

Dataset	Model	Method	Acc.	Text Tokens	Comm. Passes	AR Steps	Latency
MedQA(TW)	Qwen2.5-7B	Ours	60.20	**960**	3	**5**	**0.86**
		Latent-MAS	59.46	971	12	26	3.91
		Text-MAS	**62.88**	2092	565	580	19.85
		Single	57.64	247	-	13	0.35
		Improve	-2.68	54.1%	99.5%	99.5%	23.1×
	Qwen3-8B	Ours	**73.41**	989	3	**8**	**2.34**
		Latent-MAS	72.26	**976**	12	15	4.52
		Text-MAS	73.25	2162	589	603	29.89
		Single	65.96	336	-	97	6.01
		Improve	+0.16	54.3%	99.5%	98.7%	12.8×
	Qwen3-14B	Ours	82.76	1032	3	**102**	**8.54**
		Latent-MAS	**82.93**	**976**	12	118	18.33
		Text-MAS	81.74	2375	669	678	36.13
		Single	77.64	256	-	15	0.80
		Improve	+1.02	56.5%	99.6%	85.0%	4.2×
	Mistral-7B	Ours	46.84	1778.75	3	270	18
		Latent-MAS	**47.35**	1802	12	588	39
		Text-MAS	45.78	**4146**	982	**1018**	**63**
		Single	41.61	410	-	92	5
		Improve	+1.06	57.10	99.7%	73.48	4×
MedQA(US)	Qwen2.5-7B	Ours	45.25	**1432**	3	**7**	**1.29**
		Latent-MAS	**46.20**	1438	12	25	4.21
		Text-MAS	45.48	2752	665	676	23.09
		Single	44.27	361	-	10	0.29
		Improve	-0.23	48.0%	99.5%	99.0%	17.9×
	Qwen3-8B	Ours	71.16	**1421**	3	**14**	**3.11**
		Latent-MAS	**77.34**	1438	12	16	3.51
		Text-MAS	72.53	2552	548	561	24.70
		Single	55.12	521	-	166	10.43
		Improve	-1.37	44.3%	99.5%	97.5%	7.9×
	Qwen3-14B	Ours	69.34	1494	3	**99**	**9.44**
		Latent-MAS	68.18	**1451**	12	128	18.44
		Text-MAS	**69.99**	3091	803	812	43.98
		Single	62.22	368	-	10	0.60
		Improve	-0.65	51.7%	99.6%	87.8%	4.7×
	Mistral-7B	Ours	42.53	2135.86	3	284	17
		Latent-MAS	**43.53**	2200	12	586	39
		Text-MAS	41.15	**4957**	1045	**1134**	**61**
		Single	39.20	527	-	119	5
		Improve	+1.38	56.91	99.7%	74.96	4×

Table B.5: Main performance comparison on all models across benchmarks (continued; Part 5/6: OpenBooKQA, MMLU).

Dataset	Model	Method	Acc.	Text Tokens	Comm. Passes	AR Steps	Latency
OpenBooKQA	Qwen2.5-7B	Ours	77.60	**679**	3	**4**	**0.84**
		Latent-MAS	75.80	681	12	16	3.24
		Text-MAS	**80.40**	1646	479	486	17.02
		Single	71.42	176	-	7	0.23
		Improve	-2.80	58.7%	99.4%	99.6%	20.3×
	Qwen3-8B	Ours	86.23	**687**	3	**6**	**1.21**
		Latent-MAS	**87.61**	696	12	15	4.18
		Text-MAS	86.12	1510	395	403	14.36
		Single	85.45	219	-	50	1.78
		Improve	+0.11	54.5%	99.2%	99.0%	11.9×
	Qwen3-14B	Ours	92.73	767	3	**99**	**7.86**
		Latent-MAS	**95.85**	**705**	12	109	8.41
		Text-MAS	91.62	1592	438	446	23.41
		Single	87.14	183	-	10	0.50
		Improve	+1.11	51.8%	99.3%	77.8%	3.0×
	Mistral-7B	Ours	72.41	1325.71	3	289	11
		Latent-MAS	**73.54**	1338	12	579	25
		Text-MAS	70.65	**3066**	784	**846**	**51**
		Single	62.82	237	-	46	1
		Improve	+1.76	56.77	99.6%	65.84	5×
MMLU	Qwen2.5-7B	Ours	64.38	**946**	3	**5**	**0.88**
		Latent-MAS	**65.93**	952	12	21	2.65
		Text-MAS	64.88	1934	513	526	17.68
		Single	62.01	244	-	12	0.69
		Improve	-0.50	51.1%	99.4%	99.4%	20.1×
	Qwen3-8B	Ours	**82.79**	**956**	3	22	**2.66**
		Latent-MAS	81.47	1010	12	**16**	2.97
		Text-MAS	82.55	1887	439	455	24.75
		Single	75.34	316	-	108	4.36
		Improve	+0.24	49.3%	99.3%	95.2%	9.3×
	Qwen3-14B	Ours	83.18	**1001**	3	**27**	**3.04**
		Latent-MAS	**84.19**	1056	12	29	3.47
		Text-MAS	83.93	1899	415	471	25.75
		Single	79.58	329	-	114	4.67
		Improve	-0.75	47.3%	99.3%	94.3%	8.5×
	Mistral-7B	Ours	56.28	1422.48	3	263	17
		Latent-MAS	56.56	1439	12	583	39
		Text-MAS	**57.53**	**3862**	981	**1052**	**58**
		Single	48.95	337	-	70	4
		Improve	-1.25	63.17	99.7%	75.00	3×

Table B.6: Main performance comparison on all models across benchmarks (continued; Part 6/6: QuAIL).

Dataset	Model	Method	Acc.	Text Tokens	Comm. Passes	AR Steps	Latency
QuAIL	Qwen2.5-7B	Ours	67.07	2388	3	**9**	**1.74**
		Latent-MAS	**71.87**	**2387**	12	19	4.04
		Text-MAS	68.88	3066	312	336	11.97
		Single	65.60	599	-	7	0.32
		Improve	-1.81	22.1%	99.0%	97.3%	6.9×
	Qwen3-8B	Ours	77.44	**2378**	3	**6**	**1.49**
		Latent-MAS	77.36	2498	12	14	5.33
		Text-MAS	**78.66**	3107	311	350	12.36
		Single	72.98	612	-	17	0.68
		Improve	-1.22	23.5%	99.0%	98.9%	8.3×
	Qwen3-14B	Ours	**80.57**	2474	3	**101**	**12.70**
		Latent-MAS	79.47	**2468**	12	167	19.06
		Text-MAS	79.91	3278	431	439	24.38
		Single	77.26	608	-	8	0.59
		Improve	+0.66	24.5%	99.3%	77.0%	1.9×
	Mistral-7B	Ours	61.43	3121.59	3	296	14
		Latent-MAS	**62.94**	3139	12	580	36
		Text-MAS	61.15	**4317**	584	**620**	**37**
		Single	59.25	673	-	24	1
		Improve	+0.28	27.69	99.5%	52.26	3×

B.4 EFFICIENCY

Fig. B.2 provides a rough, back-of-the-envelope amortization analysis of Thought Blocks relative to autoregressive latent communication (Latent-MAS). Thought Blocks incur a one-time training cost, but avoid part of the repeated autoregressive latent computation required to construct inter-agent messages at inference time. Under our default four-agent setting, the estimated break-even points are approximately 8.82×10^4 task instances for the token-side cost in our accounting and 3.37×10^5 instances for latency. These estimates assume that the measured per-instance cost difference remains approximately constant as the number of inference instances grows, and should therefore be interpreted as order-of-magnitude rather than deployment-specific estimates.

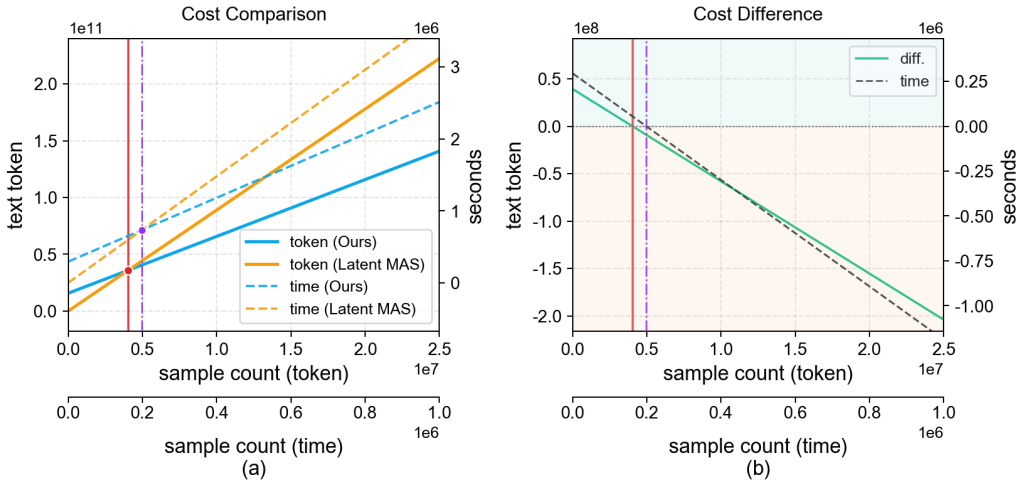


Figure B.2: Amortized token and latency cost of Thought Blocks versus autoregressive latent MAS.

For perspective, OpenAI reports that its recent Navier–Stokes workflow involved approximately 2.7 million inter-agent messages (OpenAI, 2026). While this workflow differs substantially from our experimental setting, its reported message count provides a useful reference for the communication scale that can arise in large multi-agent systems. Under our default four-agent sequential setup, each task instance contains three inter-agent transfers, so 2.7 million transfers correspond to roughly 9.0×10^5 four-agent-equivalent instances. This scale is approximately $10.2\times$ our estimated token-side break-even point and $2.7\times$ our latency break-even point. Under the same simple linear extrapolation, the one-time training overhead would account for about 9.8% of the cumulative gross token-side savings and 37.4% of the cumulative gross latency savings relative to Latent-MAS at this scale, leaving approximately 90.2% and 62.6%, respectively, after amortizing the training cost.

These numbers should not be interpreted as an estimate of the savings that would have been obtained in the OpenAI workflow itself: that system does not use Latent-MAS, and its models, communication topology, task distribution, and hardware differ from our setting. Rather, the comparison illustrates that the one-time cost of learning a direct latent communication interface can become relatively small once a system operates at hundreds of thousands to millions of inter-agent communication events. Importantly, communication at this scale is not merely hypothetical: the OpenAI Navier–Stokes workflow provides a concrete example of a real multi-agent system operating in the million-message regime. The break-even point may be reached with fewer task instances in workflows with more agents or communication rounds, provided that additional communication events yield comparable per-transfer savings from eliminating autoregressive latent rollout.

B.5 Implementation Details

Hardware Environment. All experiments are conducted on a server running CentOS Linux 7. The server is equipped with two Intel Xeon Gold 6430 CPUs, 1 TB of system memory, and eight NVIDIA RTX A6000 GPUs with 48 GB memory each.

Beacon Backbone Training. We build the compressed-context backbone following Activation Beacon (Zhang et al., 2024). During Beacon training, the language-model backbone is frozen and only the Beacon parameters are optimized, including the Beacon embedding and the Beacon-specific query, key, and value projections. Beacon positions are interleaved with the original tokens; both the sliding window and stride are set to 2,048 tokens, with full-coverage attention and access to previous Beacon states. Pretraining samples compression ratios from $\{2, 4, 8, 16, 32\}$, while fine-tuning uses $\{2, 4, 8\}$. For configuration, we use a maximum context length of 8,192 tokens, one fine-tuning epoch, bf16 precision, FlashAttention-2, and DeepSpeed ZeRO-2.

Teacher State Construction. The pre-trajectory continuation state is constructed using the same state representation as in the main method: earlier groups retain only Beacon KV states, while the final group keeps its raw-token KV states, and the layer-wise hidden states at the last raw position form the continuation seed.

For trajectory-to-state supervision, the frozen teacher performs autoregressive latent decoding from the pre-trajectory state (R_i, C_i). The teacher rollout follows a fixed decoding horizon, capped at 12 decode steps under the standard configuration, with an extended 96-step configuration used only for larger slot-count studies. Decode-stage Beacons are placed along the resulting trajectory.

After constructing the teacher rollout, the state-extraction operator Π converts the fixed-horizon latent trajectory into a fixed-size teacher continuation-memory target $\mathcal{M}_i^\star$, represented by B Beacon states. Here, B denotes the size of the distilled continuation-memory representation rather than a truncation of the teacher rollout: the teacher trajectory is generated according to its configured decoding horizon independently of the size of the resulting supervision target.

The retained states are model-native continuation states reached along the teacher's autoregressive latent trajectory. The Thought Block writer is trained to directly reconstruct the corresponding continuation-memory target from (R_i, C_i), replacing the sequential autoregressive construction of these states with a fixed-size one-shot prediction. The teacher trajectory is therefore required only for constructing supervision during training and is removed at inference time.

Thought Block Writer Training. The Thought Block writer uses a cross-attention expander with 8 attention heads. The backbone and Beacon mechanism remain frozen throughout this stage; only the writer parameters ϕ are updated. Our main configuration uses $B = 4$ learnable slots and a Beacon compression ratio of $r = 2$. We train for 10 epochs with AdamW, a learning rate of 5×10^{-5}, gradient clipping at 1.0, and dropout 0. The training loss combines KV-state matching and the auxiliary hidden-state alignment defined in Eq. 26, with weights $\lambda_{\mathrm{KV}} = 1.0$ and $\lambda_{\mathrm{aux}} = 0.5$, respectively.