INFOLAT: INFORMATIVE LATENT COMMUNICATION BETWEEN HETEROGENEOUS LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

The collaboration of heterogeneous large language models (LLMs) is an appealing way to leverage their complementary strengths. Traditional approaches communicate through text, forcing each model to compress its information-rich internal states into discrete tokens. Alternatively, latent communication offers a promising collaboration path by letting models communicate directly in a high-dimensional latent space, without explicit text. However, some existing methods only support communication between models with the same backbone; others learn a communication module for a fixed sender–receiver pair, so replacing the receiver requires retraining from scratch. Moreover, some representative methods tend to collapse the latent message into a task-agnostic one. To facilitate message understanding of heterogeneous receivers, we first represent each latent message as a sequence of latent tokens, where each latent token consists of a set of vocabulary tokens and their continuous weights. Then, we propose INFOLAT, a latent communication framework that combines latent label construction, vocabulary alignment, and decoupled sender–receiver training, enabling a sender to broadcast informative latent messages to multiple heterogeneous receivers without retraining the sender. Experiments on math and ALFWorld benchmarks demonstrate that INFOLAT consistently outperforms existing communication methods, with superior performance and shorter message sequences.

1 INTRODUCTION

Large language models (LLMs) differ in architecture (Raffel et al. (2020)), parameter scale (Kaplan et al. (2020)), and training paradigm (Ouyang et al. (2022)), resulting in significant heterogeneity in knowledge coverage, reasoning ability, and inference cost (Liang et al. (2022); Kwon et al. (2023); Wang et al. (2024)). To improve task performance and inference efficiency, communication and collaboration among heterogeneous models are often leveraged to exploit their complementary strengths (Fu et al. (2025); Yue et al. (2025); Zhang et al. (2026)). Traditional communication between heterogeneous models is text-based, requiring models to convert information-rich high-dimensional internal states into discrete token sequences (Ramesh & Li (2025); Yu et al. (2026)). Although text is highly readable, this conversion may limit precise and information-preserving transmission between models (Du et al. (2026); Liu (2026)). In this context, latent communication provides a promising path toward better model collaboration by enabling LLMs to communicate in a high-dimensional latent space without explicit text (Fu et al. (2026); Zou et al. (2026b)).

Existing latent communication methods can be divided into two categories: training-free (Pham et al. (2024); Ramesh & Li (2025); Peng et al. (2026); Shi et al. (2026); Zou et al. (2026b)) and training-based (Zheng et al. (2025); Zou et al. (2026a); Du et al. (2026); Fu et al. (2026)), as illustrated in Figure 1. Training-free methods enable latent communication by exchanging token embeddings (Pham et al. (2024)), hidden states (Ramesh & Li (2025); Peng et al. (2026)), or KV caches (Shi et al. (2026); Zou et al. (2026b)), but typically rely on a shared backbone, limiting their ability to exploit the complementary strengths of heterogeneous models. Although training-based methods can enable communication among heterogeneous models, they usually learn a communication module for a specific sender–receiver pair (Du et al. (2026); Fu et al. (2026); Zou et al. (2026a)). Replacing the receiver therefore requires retraining the entire communication module from scratch.

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

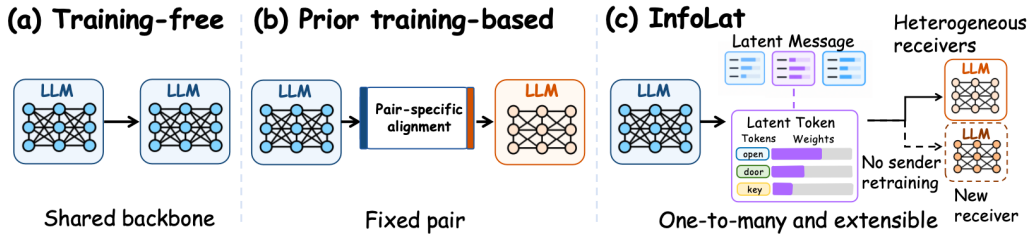


Figure 1: Comparison of latent communication paradigms: training-free methods typically rely on a shared backbone; training-based methods learn a communication module for a fixed sender–receiver pair; INFOLAT enables a single sender to broadcast to multiple heterogeneous receivers.

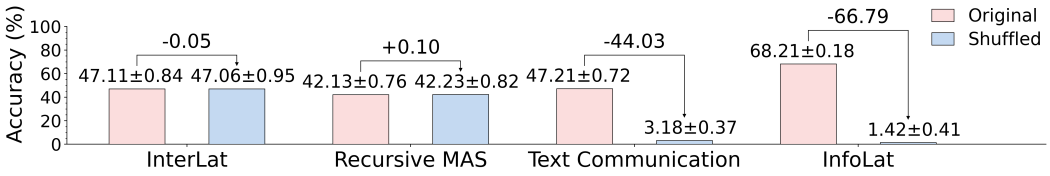


Figure 2: Performance degradation of different latent communication methods when given a shuffled latent message from other problems. The mild degradation suggests that these latent messages carry little task-specific information, so the receiver relies on them weakly. (Appendix D.1 for details).

Moreover, we observe that existing training-based methods (Du et al. (2026); Zou et al. (2026a)) tend to collapse the latent message into a task-agnostic message that carries little task-specific information. To verify this, we keep the receiver's problem input unchanged and randomly shuffle the latent messages corresponding to different problems, finding that the tested methods achieve nearly unchanged performance, as shown in Figure 2. This phenomenon indicates that the communication in these methods collapses and fails to convey information as intended.

To address these challenges, we represent each latent message as a sequence of latent tokens, where each latent token consists of a set of vocabulary tokens and their continuous weights. Each model constructs a latent token embedding as a weighted combination of the corresponding vocabulary token embeddings. We further align each receiver's vocabulary with the sender's vocabulary, allowing receivers with different vocabularies and hidden dimensions to construct latent token embeddings in their own input embedding spaces. Specifically, we propose INFOLAT, a latent communication framework for heterogeneous models: (1) since no ground-truth latent message exists, we first introduce an encoder-decoder architecture that learns to compress text-formed messages into latent labels, thereby providing explicit supervision for the communication content; (2) then, we train the sender to generate latent messages given the task information; (3) finally, we train the receiver to complete the task given the task information and latent message, thereby decoupling the training of the sender and the receiver. As a result, messages generated by the same sender can be broadcast to multiple heterogeneous receivers, and a new receiver can be added without retraining the sender.

In summary, the main contributions of this paper are as follows. (1) We identify two limitations of existing latent communication methods: the learned communication module is bound to fixed sender-receiver pairs, which prevents the sender from being reused across different receivers, and the latent messages tend to collapse into task-agnostic messages. (2) We propose INFOLAT, a latent communication framework for heterogeneous models that, through latent label construction, vocabulary alignment and decoupled sender–receiver training, enables a single sender to broadcast informative messages to multiple heterogeneous receivers without retraining the sender. (3) Extensive experiments demonstrate that, averaged across cross-scale and cross-family settings, INFOLAT achieves relative improvements of 13.9% in average pass@1 on math benchmarks and 7.2% in average success rate on ALFWorld over the strongest communication baselines.

2 RELATED WORK

Latent Space Representation. Existing work explores latent space representations for information compression and reasoning. Information compression methods encode textual contexts into

compact continuous representations to reduce storage and computation costs. Gist Tokens (Mu et al., 2023) and AutoCompressors (Chevalier et al., 2023) summarize long contexts into reusable representations; ICAE (Ge et al., 2024) learns compact memory slots through reconstruction and language modeling; and 500xCompressor (Li et al., 2025) stores context information in compressed KV states to support higher compression ratios. These studies provide a foundation for designing compact latent messages that preserve task-relevant information. Beyond storing compressed information, latent representations can also support intermediate reasoning computation. Latent reasoning methods represent reasoning steps as continuous states (Hao et al., 2025; Cheng & Van Durme, 2024; Shen et al., 2025; Wu et al., 2025) or soft token embeddings (Zhang et al., 2025b; Deng et al., 2025). These approaches aim to improve the reasoning capabilities and efficiency of a single model by reducing reliance on explicit textual reasoning chains. They don't address the alignment of vocabularies and representation spaces required for latent communication between heterogeneous models.

Latent Communication. Existing latent communication methods can be broadly categorized as training-free or training-based. Training-free methods exchange different forms of continuous messages: CIPHER (Pham et al., 2024) uses probability-weighted token embeddings; Ramesh & Li (2025) combine intermediate hidden activations; StateBridge (Peng et al., 2026) aligns message hidden states with input embeddings through a closed-form transformation; and KVComm (Shi et al., 2026) and LatentMAS (Zou et al., 2026b) communicate through KV caches. Their applicability is constrained by model compatibility: StateBridge assumes agents share the same pretrained model, KVComm and LatentMAS require compatible backbone architectures for cache sharing, and CIPHER permits different models but requires a shared tokenizer. These requirements limit communication across heterogeneous models with different vocabularies and internal representations. Training-based methods relax these constraints by learning communication module between models. ThoughtComm (Zheng et al., 2025), RecursiveMAS (Zou et al., 2026a), and Interlat (Du et al., 2026) enable communication through hidden states. ThoughtComm injects latent thoughts as prefixes, RecursiveMAS generates and transfers latent states, and Interlat learns adapters for hidden-state messages and generators for compressed latent messages. Cache-to-Cache (Fu et al., 2026) instead operates on KV caches, projecting and fusing them across models. Despite supporting heterogeneous models, these approaches rely on communication modules optimized for specific sender–receiver pairs, limiting their reuse across receivers. Beyond model compatibility and pair-specific training, some methods (Du et al. (2026); Zou et al. (2026b)) also suffer from communication collapse, where the learned messages fail to convey task-specific information. INFOLAT overcomes limitations by decoupling sender and receiver training and supervising communication content, enabling a sender to broadcast to heterogeneous receivers without retraining the sender.

3 METHODOLOGY

We first give an overview of INFOLAT in Figure 3 and then detail components in the subsections.

3.1 PROBLEM FORMULATION

In this subsection, we will first introduce the communication paradigm between two models. Then, we will formalize the message generation and message transmission processes for both text and latent communication.

Communication Paradigm. Typically, a communication process can be abstracted as a collection of sender–receiver interactions. Formally, we denote the sender and receiver as S_θ and R_ϕ, where θ and ϕ are their respective model parameters. We use p_θ and p_ϕ to denote the conditional predictive distributions induced by S_θ and R_ϕ, respectively. Following previous work (Carmeli et al. (2023); Wang et al. (2025); Zhang et al. (2025a); Du et al. (2026)), the sender first generates a message M from the task input x and transmits it to the receiver. The receiver produces an output y conditioned on the same task input x and the received message M. This process can be formulated as $M = S_\theta(x), y = R_\phi(x, M)$.

We denote the sender's and receiver's tokenizers as τ_S and τ_R, respectively. The sender's vocabulary is $\mathcal{V}_S = \{v_i\}_{i=1}^{|\mathcal{V}_S|}$, where v_i denotes its i-th token; the receiver's vocabulary is denoted by $\mathcal{V}_R$. The input embedding maps $\mathbf{e}_S : \mathcal{V}_S \to \mathbb{R}^{d_S}$ and $\mathbf{e}_R : \mathcal{V}_R \to \mathbb{R}^{d_R}$ assign each token its model-specific

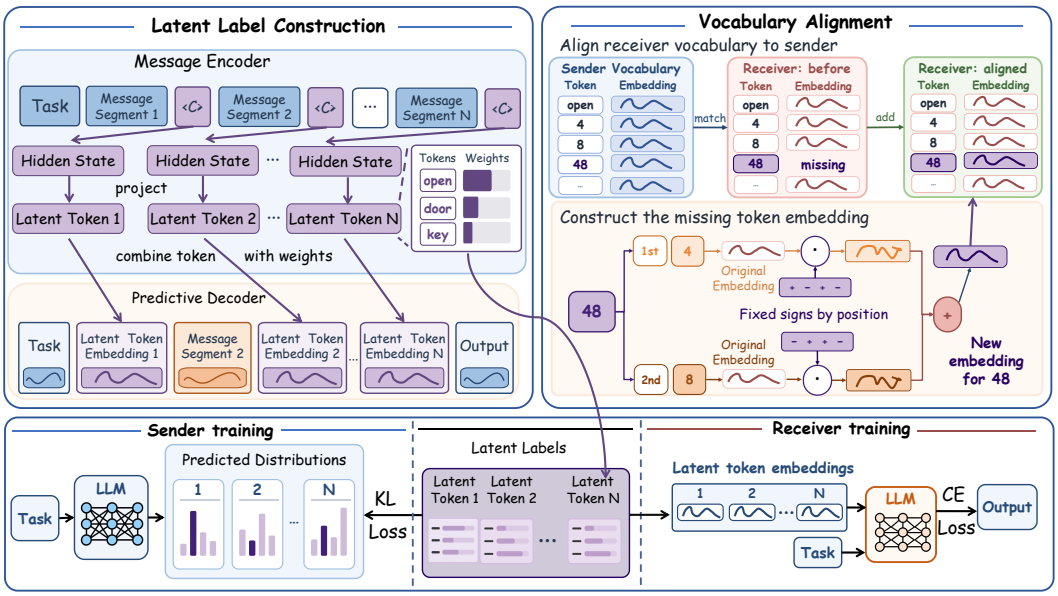


Figure 3: Overview of INFOLAT. (1) Latent Label Construction: an encoder–decoder compresses reference text messages into latent labels, providing supervision for communication content. (2) Vocabulary Alignment: the receiver's vocabulary is aligned with the sender's, enabling heterogeneous models to reconstruct latent tokens in their own embedding spaces. (3) Sender Training: the sender is trained to generate latent messages from the task input by matching the latent labels via a KL divergence loss. (4) Receiver Training: the receiver is trained to predict the task output from the task input and the latent message.

input embedding, where d_S and d_R are the sender's and receiver's hidden dimensions, respectively. The sender tokenizes the task input as $x^S = \tau_S(x) = (x_1^S, \ldots, x_{T_S}^S)$, where $x_j^S \in \mathcal{V}_S$ is a token and T_S is the number of sender input tokens.

Text Communication. At generation step t, the sender S_θ processes x^S and previously generated message tokens $(u_1, \ldots, u_{t-1})$. We denote the last-layer hidden state at the final position of this prefix by $\mathbf{h}_t^S \in \mathbb{R}^{d_S}$. The sender then projects $\mathbf{h}_t^S$ to scores over the tokens in $\mathcal{V}_S$ through the language modeling head and selects a token $u_t \in \mathcal{V}_S$ by sampling or greedy decoding(Pham et al. (2024); Feng et al. (2024)). The embedding $\mathbf{e}_S(u_t)$ is fed back into S_θ for the next generation step, and this process continues until message generation terminates. The L generated tokens $(u_1, \ldots, u_L)$ are decoded into the text message M^{text}, which is then transmitted to the receiver. The receiver tokenizes M^{text} using τ_R and maps the resulting tokens to input embeddings through $\mathbf{e}_R$. These message embeddings are used together with the task input x to generate y (Ramesh & Li (2025).

Latent Communication. In this work, a message is composed of N latent tokens: $M^{\text{latent}} = (M_1^{\text{latent}}, \ldots, M_N^{\text{latent}})$, $M_t^{\text{latent}} = \{(v_{t,i}, \alpha_{t,i})\}_{i=1}^K$, where each latent token is represented by K vocabulary tokens $v_{t,i}$ and non-negative weights $\alpha_{t,i}$ that sum to one. The sender and receiver map each latent token to an embedding using their respective input embedding maps: $\mathbf{z}_t = \sum_{i=1}^K \alpha_{t,i}\, \mathbf{e}(v_{t,i})$. The sender S_θ generates the message autoregressively. At step t, it processes x^S together with the preceding latent token embeddings $\mathbf{z}_{<t}^S$ to obtain the last-layer hidden state $\mathbf{h}_t^S$ at the final prefix position. Following Pham et al. (2024), it scores the vocabulary tokens in $\mathcal{V}_S$ by projecting $\mathbf{h}_t^S$ onto their embeddings, selects the top-K tokens, and normalizes their scores to obtain the weights $\alpha_{t,i}$. The sender then maps this latent token to $\mathbf{z}_t^S$ and appends it to the input prefix for the next step. Generation repeats until termination. Upon receiving M^{latent}, the receiver constructs $\mathbf{z}_{1:N}^R$ using its own embedding map $\mathbf{e}_R$ and generates y conditioned on x and $\mathbf{z}_{1:N}^R$.

3.2 LATENT LABEL CONSTRUCTION

To train the sender to generate latent messages, we first need to construct latent labels as supervision signals, which are not directly available in text communication data. We therefore introduce a message encoder and a predictive decoder to construct latent labels M^{label} from reference text messages M^{text}. The message encoder shares the backbone architecture with the sender and compresses each

message segment into a latent token, represented as a set of token–weight pairs. The predictive decoder uses the corresponding latent token embeddings to predict subsequent message segments and the task output, providing supervision that encourages the encoder to retain information needed for message continuation and task completion.

Message Encoder Design. Using the token sequence $(u_1, \ldots, u_L)$ of M^{text}, we place a segment boundary every ρ tokens. The t-th text segment B_t corresponds to $(u_{(t-1)\rho+1}, \ldots, u_{t\rho})$, yielding $N = \lceil L/\rho \rceil$ segments. A learnable compression token $\langle \text{c} \rangle$ is inserted after each segment, yielding the encoder input $[x, B_1, \langle \text{c} \rangle, B_2, \langle \text{c} \rangle, \ldots, B_N, \langle \text{c} \rangle]$. The encoder assigns consecutive position indices to this sequence, with each inserted compression token occupying one position. To ensure that message information can flow into the compression tokens, we adopt a modified causal attention mask. Specifically, each compression token can attend to all preceding text tokens. All subsequent tokens, including later compression tokens, are prevented from attending to it. The last-layer hidden state of the t-th $\langle \text{c} \rangle$ (i.e., the one following B_t) is converted into a latent token M_t^{label}, which serves as the supervision signal. All N labels are computed in a single encoder forward pass (Mu et al. (2023); Chevalier et al. (2023)).

Predictive Decoder Design. The decoder constructs a latent token embedding $\mathbf{z}_t$ from each M_t^{label} by combining the corresponding token embeddings with the label weights. To supervise the information retained in these embeddings, we place each $\mathbf{z}_t$ before the next text segment B_{t+1} and place the task output after the final latent token embedding: $[x, \mathbf{z}_1, B_2, \mathbf{z}_2, \ldots, B_N, \mathbf{z}_N, y]$. To make the decoder rely on latent token embeddings for information from earlier message segments, we design a modified causal attention mask. Specifically, each position containing a latent token embedding $\mathbf{z}_t$ can attend only to x and $\mathbf{z}_{1:t}$. When predicting a token in B_{t+1}, the decoder can access only x, $\mathbf{z}_{1:t}$, and preceding tokens within B_{t+1}. The output y is similarly predicted using only x, all latent token embeddings, and preceding output tokens.

The interleaved text segments serve as prediction targets, while the latent token embeddings provide the compressed context. We therefore reassign the position indices so that successive latent token embeddings receive consecutive indices, regardless of how many text tokens separate them. If $\mathbf{z}_t$ has position index π_t, the next latent token embedding $\mathbf{z}_{t+1}$ receives $\pi_t + 1$. The tokens in B_{t+1} are indexed as a continuation of the latent prefix, with indices $\pi_t + 1, \pi_t + 2, \ldots$. The output tokens are indexed after the final latent token embedding, starting at $\pi_N + 1$.

We supervise subsequent message prediction and task output prediction with a token-level cross-entropy loss. Writing $B_{N+1} := y$ for the task output, the loss is

$$\mathcal{L}_{\text{label}} = -\frac{1}{N_{\text{sup}}} \sum_{t=1}^{N} \log p_{\text{dec}}(B_{t+1} \mid x, \mathbf{z}_{1:t}), \tag{1}$$

where p_{dec} denotes the decoder's predictive distribution and N_{sup} is the total number of supervised decoder tokens.

Jointly training the encoder and the decoder from scratch is unstable, since the encoder initially emits uninformative token distributions. Within the latent label construction stage, we therefore optimize the two components in three steps: we first freeze the decoder and train only the encoder with $\mathcal{L}_{\text{label}}$; we then freeze the encoder and train only the decoder with the same loss; finally, we unfreeze both components and jointly fine-tune them. After training, we apply the encoder to (x, M^{text}) and store the resulting latent labels M^{label} for sender training. The predictive decoder only defines which information the labels should retain and is discarded after label construction.

3.3 Vocabulary Alignment

Unlike text communication, where the message M^{text} is transmitted as text, latent communication transmits M^{latent} as a sequence of latent tokens, each represented as a set of token–weight pairs. The sender and receiver may use different vocabularies, and a token selected by the sender may be absent from the receiver's vocabulary. To enable the receiver to construct latent token embeddings from the transmitted message, we align the receiver's vocabulary with the sender's. The aligned vocabularies are used only for latent communication, while the models retain their original vocabularies for regular text processing and generation.

For each non-special token $v_i \in \mathcal{V}_S$, we search $\mathcal{V}_R$ for a token with the same text. For tokens already present, we use their existing receiver input embeddings. For a missing token, we tokenize

its text with the receiver's tokenizer into $(u^R_{i,1}, \ldots, u^R_{i,\ell_i})$ and construct the corresponding embedding $\mathbf{e}_R(v_i)$ from these token embeddings.

To preserve token order when composing multiple embeddings, we apply a fixed sign transformation at each token position before summation: $\mathbf{e}_R(v_i) = \sum_{j=1}^{\ell_i} \mathbf{r}_j \odot \mathbf{e}_R(u^R_{i,j})$, where $\odot$ denotes elementwise multiplication and each $\mathbf{r}_j$ has entries independently sampled uniformly from $\{-1, +1\}$ and is kept fixed across vocabulary tokens (Heddes et al. (2024)).

To match the scale and direction of native receiver embeddings, we further calibrate $\mathbf{e}_R(v_i)$ (more datails in Appendix B.1), and extend the input embedding map $\mathbf{e}_R$ to assign it the constructed embedding (Diehl Martinez et al. (2024)). Both vocabularies and all associated embeddings remain fixed throughout training.

3.4 Sender Training

With the constructed latent labels M^{label}, we train the sender to generate latent messages from the task input. We minimize the KL divergence between this label distribution and the sender's predicted distribution. Specifically, we train the sender with teacher forcing. At step t, the sender receives x and the latent token embeddings $\mathbf{z}^S_{<t}$ constructed from the preceding labels using its input token embeddings. The sender's predicted distribution $p_\theta(\cdot \mid x, \mathbf{z}^S_{<t})$ is normalized over the full vocabulary $\mathcal{V}_S$. The KL loss is defined as

$$\mathcal{L}_{\text{KL}} = \frac{1}{N} \sum_{t=1}^{N} D_{\text{KL}}\big(M^{\text{label}}_t \,\|\, p_\theta(\cdot \mid x, \mathbf{z}^S_{<t})\big)\,. \tag{2}$$

We update the sender while keeping its vocabulary and the associated embeddings fixed. At inference time, the sender generates the latent message M^{latent} autoregressively from x, using the embeddings of its own previously generated latent tokens as context.

3.5 Receiver Training

We keep the sender fixed and train the receiver to interpret latent messages for task output generation. During training, we condition the receiver on the task input x and the constructed latent labels M^{label}. For each M^{label}_t, the receiver constructs the latent token embedding $\mathbf{z}^R_t$ using its input embedding map $\mathbf{e}_R$ over the aligned vocabulary $\mathcal{V}_R$. We train the receiver with teacher forcing to predict the task output y, minimizing the following cross-entropy (CE) loss:

$$\mathcal{L}_{\text{CE}} = -\frac{1}{L_y} \sum_{j=1}^{L_y} \log p_\phi\big(y_j \mid x, \mathbf{z}^R_{1:N}, y_{<j}\big)\,. \tag{3}$$

We fine-tune the receiver while keeping its input token embeddings fixed. The training processes of sender and receiver are decoupled. Hence, the same latent message can be broadcast to multiple receivers, each constructing latent token embeddings using its own input embedding map over its aligned vocabulary and independently generating the task output. InfoLAT thus serves as a shared protocol, rather than a private conversation between two specific models.

4 Experiments

4.1 Experiments Setups

Tasks and Datasets. Following previous work on latent communication (Du et al. (2026)), we evaluate InfoLAT on two types of tasks: a set of math benchmarks and an embodied interactive benchmark, ALFWorld (Shridhar et al., 2021). For the math benchmarks, we train InfoLAT on GSM8K-aug (Cobbe et al., 2021) and evaluate on GSM8K-aug, GSM-Hard (Gao et al., 2023), MultiArith (Roy & Roth, 2015), and SVAMP (Patel et al., 2021). For ALFWorld, we train InfoLAT on exactly the same training set as Interlat, which is collected by Song et al. (2024), and evaluated on the official seen and unseen test splits. We report more details in Appendix A.1.

Evaluation Metrics. Following Du et al. (2026), we report *pass@1* for the math benchmarks and the task success rate for ALFWorld. We also report the message length $\#L$, defined as the average number of tokens (or latent tokens) per message, with text counted using the receiver's tokenizer;

Table 1: Math benchmark results for single models and model communication methods. Best and second-best results in each block column are marked in **bold red** and underlined blue.

Model / Method	GSM8K-aug		GSM-Hard		SVAMP		MultiArith		Average	
	Pass@1 ↑	$\#L$ ↓	Pass@1 ↑	$\#L$ ↓	Pass@1 ↑	$\#L$ ↓	Pass@1 ↑	$\#L$ ↓	Pass@1 ↑	$\#L$ ↓
No Communication										
Llama-3.2-3B-Instruct	56.31 ± 0.77	25.88	18.40 ± 0.30	35.96	72.67 ± 1.51	11.57	97.78 ± 0.63	14.71	61.29 ± 0.46	22.03
Llama-3.2-1B-Instruct	38.06 ± 1.44	26.59	9.10 ± 0.34	35.12	53.44 ± 1.15	12.60	93.15 ± 2.38	14.80	48.44 ± 0.76	22.28
Qwen2.5-1.5B-Instruct	42.51 ± 1.33	32.11	15.37 ± 0.39	51.58	68.56 ± 2.93	15.37	94.81 ± 0.36	18.09	55.31 ± 0.82	29.29
Cross-Scale Model Communication: Llama-3.2-3B-Instruct → Llama-3.2-1B-Instruct										
TEXT COMMUNICATION	47.21 ± 0.72	27.31	13.82 ± 0.39	36.83	55.44 ± 0.58	12.97	96.85 ± 0.73	14.78	53.33 ± 0.31	22.97
INTERLAT	47.11 ± 0.84	15.00	11.75 ± 0.15	**15.00**	62.56 ± 0.69	15.00	95.56 ± 0.00	15.00	54.25 ± 0.27	15.00
RECURSIVEMAS	42.13 ± 0.76	15.00	10.92 ± 0.08	**15.00**	64.00 ± 0.58	15.00	94.07 ± 1.28	15.00	52.78 ± 0.40	15.00
SHUFFLED	1.42 ± 0.41	**12.93**	0.10 ± 0.20	15.39	1.11 ± 0.58	**6.05**	3.15 ± 0.73	**7.16**	1.45 ± 0.26	**10.38**
INFOLAT	**68.21** ± 0.18	**12.93**	**16.12** ± 0.10	15.39	**69.89** ± 0.58	**6.05**	**98.33** ± 0.00	**7.16**	**63.14** ± 0.15	**10.38**
Cross-Family Model Communication: Llama-3.2-3B-Instruct → Qwen2.5-1.5B-Instruct										
TEXT COMMUNICATION	50.32 ± 0.87	34.70	16.91 ± 0.31	55.16	65.78 ± 1.57	16.82	97.41 ± 0.96	18.34	57.61 ± 0.51	31.26
INTERLAT	48.93 ± 1.25	15.00	15.37 ± 0.90	**15.00**	70.44 ± 1.35	15.00	94.81 ± 0.64	15.00	57.39 ± 0.54	15.00
RECURSIVEMAS	44.07 ± 0.61	15.00	15.42 ± 0.38	**15.00**	70.33 ± 0.33	15.00	97.59 ± 0.32	15.00	56.85 ± 0.21	15.00
SHUFFLED	1.57 ± 0.52	**12.93**	0.25 ± 0.13	15.39	1.44 ± 1.09	**6.05**	2.41 ± 1.58	**7.16**	1.42 ± 0.50	**10.38**
INFOLAT	**68.81** ± 0.25	**12.93**	**17.54** ± 0.22	15.39	**72.44** ± 0.44	**6.05**	**97.78** ± 0.00	**7.16**	**64.14** ± 0.14	**10.38**

for the NO COMMUNICATION setting, it instead denotes the length of the model's own reasoning chain. Each latent token occupies one sequence position, just as a text token does; $\#L$ effectively reflects communication cost (Appendix A.5).

Training and Implementation Details. For the math benchmarks, we adopt the configuration: Llama-3.2-3B-Instruct (Grattafiori et al., 2024) as the sender, and Llama-3.2-1B-Instruct (Grattafiori et al., 2024) and Qwen2.5-1.5B-Instruct (Yang et al., 2024) as the receivers. Since ALFWorld is more complex and harder than the math benchmarks, we adopt the most widely used model configuration for this benchmark, which uses larger models: Qwen2.5-7B-Instruct (Yang et al., 2024) as the sender, and Llama-3.2-1B-Instruct (Grattafiori et al., 2024) and Qwen2.5-1.5B-Instruct as the receivers. Full training configurations and inference protocols are provided in Appendix A.3.

Baselines. We first evaluate each model independently without any communication, denoted NO COMMUNICATION, to establish its base reasoning performance (More details in Appendix A.2). All subsequent baselines are initialized from the models fine-tuned under this NO COMMUNICATION setting. We compare INFOLAT against three groups of baselines. First, we include TEXT COMMUNICATION as a reference for text-mediated collaboration, in which the sender generates the message as plain text and the receiver conditions on this textual message. Second, we compare with two representative latent communication methods, INTERLAT (Du et al., 2026) and RECURSIVEMAS (Zou et al., 2026a), both of which transmit continuous hidden states between agents; following their original settings, we fix the messages they send to a constant length. Third, we include the SHUFFLED baseline, where each receiver is fed a latent message generated from a different task, which verifies that INFOLAT indeed exploits the task-specific information encoded in the latent message. We report the detailed settings of all baselines in Appendix A.4.

4.2 MAIN RESULTS

As presented in Tables 1 and 2, INFOLAT consistently achieves the best performance among all evaluated communication methods, across the cross-scale and cross-family settings, and on both the math and ALFWorld benchmarks.

On the math benchmarks, INFOLAT achieves relative improvements of 16.4% and 11.3% in average pass@1 over the strongest communication baselines, INTERLAT and TEXT COMMUNICATION, in the cross-scale and cross-family settings, respectively, while reducing average message length by 30.8% and 66.8% relative to those same baselines.

For ALFWorld, INFOLAT improves success rate by 10.2% and 4.1% relative to the strongest communication baselines, TEXT COMMUNICATION and INTERLAT, in the cross-scale and cross-family settings, respectively. The corresponding average message lengths are 50.8% and 81.1% shorter. These results demonstrate the effectiveness of INFOLAT across both communication settings and tasks. Shuffling the latent messages significantly degrades task performance across both benchmarks and communication settings. These results indicate that INFOLAT enables latent messages to convey task-specific information that receivers use for task completion.

Table 2: ALFWorld results for single models and model communication methods. Best and second-best results in each block column are marked in **bold red** and underlined blue.

Model / Method	Seen		Unseen	
	Acc. ↑	#L ↓	Acc. ↑	#L ↓
No Communication				
Llama-3.2-1B-Instruct	68.10 ± 2.18	25.01	59.45 ± 1.14	25.08
Qwen2.5-1.5B-Instruct	64.29 ± 2.14	23.73	62.44 ± 3.45	24.37
Qwen2.5-7B-Instruct	71.19 ± 3.22	25.27	66.92 ± 4.74	25.31
Cross-Scale Model Communication: Qwen2.5-7B-Instruct → Qwen2.5-1.5B-Instruct				
TEXT COMMUNICATION	60.71 ± 0.71	24.05	61.69 ± 1.14	24.80
INTERLAT	60.71 ± 1.43	64.00	58.21 ± 4.54	64.00
RECURSIVEMAS	61.43 ± 2.58	64.00	56.72 ± 1.49	64.00
SHUFFLED	15.00 ± 1.43	16.10	13.93 ± 1.14	16.37
INFOLAT	**64.76** ± 1.09	**11.63**	**70.15** ± 4.16	**12.38**
Cross-Family Model Communication: Qwen2.5-7B-Instruct → Llama-3.2-1B-Instruct				
TEXT COMMUNICATION	67.62 ± 0.41	24.27	62.69 ± 2.69	24.18
INTERLAT	66.90 ± 1.80	64.00	63.93 ± 1.14	64.00
RECURSIVEMAS	49.29 ± 5.00	64.00	45.77 ± 3.68	64.00
SHUFFLED	5.71 ± 0.71	16.06	5.72 ± 2.62	16.34
INFOLAT	**69.52** ± 1.49	**11.85**	**66.67** ± 3.02	**12.31**

4.3 ABLATION STUDY

To validate the design of INFOLAT, we examine the three-step optimization of the latent label construction stage, vocabulary alignment, latent message representation, and receiver training (Table 3).

To assess this optimization schedule, we compare three variants: *w/o decoder-only* skips decoder-only adaptation before joint training; *encoder only* trains only the encoder with a frozen decoder; and *joint only* jointly trains the encoder and decoder without the separate encoder-only and decoder-only steps. These variants reduce average pass@1 by 1.2%, 5.8%, and 4.8% relative to the full method, respectively, supporting the complete three-step schedule for constructing effective latent labels.

We first remove vocabulary alignment (*w/o alignment*), retaining the sender's vocabulary while using the receiver's native vocabulary to interpret latent messages. This reduces average pass@1 by 3.0% relative to the full method. We also test an alternative alignment strategy, *intersection*, which allows the sender to use only tokens that have a matching token in the receiver's vocabulary. It incurs a 9.9% relative drop in average pass@1, likely because the restricted vocabulary limits sender expressiveness by excluding tokens available to the sender but absent from the shared vocabulary. This strategy also limits scalability: adding a heterogeneous model can shrink the intersection, requiring the communication vocabulary to be rebuilt and the affected models to be retrained.

Table 3: Ablation results on math benchmarks with Llama-3.2-3B-Instruct as the sender and Qwen2.5-1.5B-Instruct as the receiver.

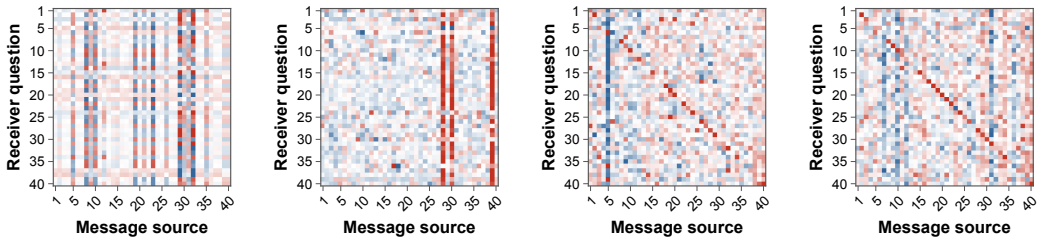
Variant	GSM8K-aug	GSM-Hard	SVAMP	MultiArith	Average
INFOLAT	**68.81** ± 0.25	**17.54** ± 0.22	**72.44** ± 0.44	97.78 ± 0.00	**64.14**
Latent Label Construction					
w/o decoder-only	68.17 ± 0.09	15.77 ± 0.06	71.89 ± 0.16	97.78 ± 0.00	63.39
encoder only	65.43 ± 0.16	15.69 ± 0.16	66.44 ± 0.42	94.07 ± 0.26	60.41
joint only	62.22 ± 0.18	14.68 ± 0.16	68.33 ± 0.33	**98.89** ± 0.00	61.03
Vocabulary Alignment					
w/o alignment	63.20 ± 0.09	17.36 ± 0.27	70.56 ± 0.51	97.78 ± 0.56	62.23
intersection	59.77 ± 0.07	13.77 ± 0.04	59.89 ± 0.16	97.78 ± 0.00	57.80
Latent Message Representation					
hidden state	42.43 ± 0.64	12.28 ± 0.16	60.00 ± 0.47	66.85 ± 0.52	45.39
Receiver Training					
w/o receiver training	17.36 ± 0.53	5.13 ± 0.34	48.22 ± 0.68	32.04 ± 1.72	25.69

To assess our latent message design, we represent each latent token using the sender's last-layer hidden state (*hidden State*) instead of vocabulary tokens and their continuous weights.

We learn a communication module, implemented as a projector, to map these hidden states into the receiver's input embedding space. This variant reduces average pass@1 by 29.2% relative to the full method, supporting our latent message design. Moreover, the learned communication module is bound to a specific sender–receiver pair and must be retrained when the receiver is replaced.

Finally, we omit receiver training (*w/o receiver training*). Average pass@1 drops by 59.9% relative to the full method, highlighting that it is still necessary to adapt the receiver to latent messages.

4.4 Additional Analysis



(a) InterLat (b) RecursiveMAS (c) Text Communication (d) InfoLat

Figure 4: Question–message compatibility on 40 randomly sampled GSM8K-aug test problems, with Llama-3.2-3B-Instruct as the sender and Qwen2.5-1.5B-Instruct as the receiver. Rows denote receiver questions and columns denote message sources. Red indicates higher compatibility; diagonal entries correspond to matched question–message pairs.

Task-Specific Information in Messages. We examine whether the receiver uses task-specific information carried by latent messages. For each question–message pair, we compute a *compatibility score*, which measures the relative likelihood that the receiver assigns to the task output when conditioned on the message (see Appendix D.2). As shown in Figure 4, InterLat and RecursiveMAS show no clear diagonal structure, indicating little systematic advantage for matched messages. Together with the message-shuffling results, this observation supports our diagnosis of communication collapse in these baselines. In contrast, InfoLat exhibits a clear diagonal pattern, similar to Text Communication: messages from the same question generally yield higher compatibility scores than those from other questions. These results suggest that the latent messages generated by InfoLat carry task-specific information that the receiver uses for task output generation.

Message Representation. Some methods use hidden-state messages (Du et al., 2026; Zou et al., 2026a). We compare this design with our representation based on vocabulary tokens and their continuous weights, using the *Hidden State* variant in Table 3. InfoLat quickly reaches a stable 69.5% GSM8K-aug accuracy, versus a 44.8% peak for hidden states (Figure 5); further training does not close the gap. These results support our latent message design, which facilitates receiver training and achieves better task performance at inference (More Details in appendix D.4) .

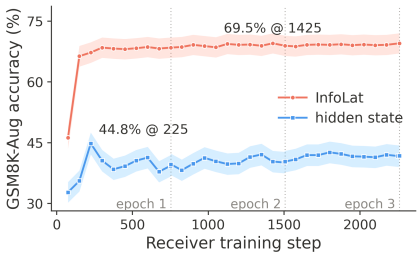


Figure 5: InfoLat vs. hidden states on GSM8K-aug (Llama → Qwen).

5 Conclusion

We studied latent communication between heterogeneous LLMs and identified two limitations of existing methods: the communication module is bound to a fixed sender–receiver pair, and latent messages tend to collapse into task-agnostic messages. To overcome these limitations, we introduce InfoLat with latent label construction, vocabulary alignment and decoupled sender–receiver training, enabling one trained sender serve multiple heterogeneous receivers. Experiments on math benchmarks and ALFWorld demonstrate that InfoLat outperforms prior methods while conveying task-specific information. In future work, we plan to extend InfoLat to multi-sender multi-receiver scenarios and a wider range of model families.

AI USE STATEMENT

We used generative AI tools solely to polish author-written text, including correcting grammar, refining wording, and improving clarity and readability. These tools were not used to generate substantive sections of the paper, conduct literature searches, develop research ideas or methodology, implement methods, generate or analyze data, interpret results, or formulate or prove mathematical claims. All AI-assisted edits were reviewed by the authors to ensure that they accurately preserved the intended meaning and technical content. We take full responsibility for the final content of this work, including all text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Boaz Carmeli, Ron Meir, and Yonatan Belinkov. Emergent quantized communication. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2023.

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

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

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.

Jingcheng Deng, Liang Pang, Zihao Wei, Shicheng Xu, Zenghao Duan, Kun Xu, Yang Song, Huawei Shen, and Xueqi Cheng. Llm latent reasoning as chain of superposition. *arXiv preprint arXiv:2510.15522*, 2025.

Richard Diehl Martinez, Zébulon Goriely, Andrew Caines, Paula Buttery, and Lisa Beinborn. Mitigating frequency bias and anisotropy in language model pre-training with syntactic smoothing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

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. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026.

Zijian Feng, Hanzhang Zhou, Zixiao Zhu, Junlang Qian, and Kezhi Mao. Unveiling and manipulating prompt influence in large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

Jiale Fu, Yuchu Jiang, Junkai Chen, Jiaming Fan, Xin Geng, and Xu Yang. Fast large language model collaborative decoding via speculation. In *Proceedings of the International Conference on Machine Learning (ICML)*, 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. In *International Conference on Learning Representations (ICLR)*, 2026.

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

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. In *International Conference on Learning Representations (ICLR)*, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 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. In *Proceedings of the Conference on Language Modeling (COLM)*, 2025.

Mike Heddes, Igor Nunes, Tony Givargis, Alexandru Nicolau, and Alex Veidenbaum. Hyperdimensional computing: a framework for stochastic computation and symbolic AI. *Journal of Big Data*, 2024.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th symposium on operating systems principles*, 2023.

Zongqian Li, Yixuan Su, and Nigel Collier. 500xcompressor: Generalized prompt compression for large language models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*, 2022.

Yingzhuo Liu. Beyond tokens: a unified framework for latent communication in LLM-based multi-agent systems. *arXiv preprint arXiv:2606.05711*, 2026.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP models really able to solve simple math word problems? In *The Nations of the Americas Chapter of the Association for Computational Linguistics*, 2021.

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 *Proceedings of the Conference on Language Modeling (COLM)*, 2026.

Chau Pham, Boyi Liu, Yingxiang Yang, Zhengyu Chen, Tianyi Liu, Jianbo Yuan, Bryan A. Plummer, Zhaoran Wang, and Hongxia Yang. Let models speak ciphers: Multiagent debate through embeddings. In *International Conference on Learning Representations (ICLR)*, 2024.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 2020.

Vignav Ramesh and Kenneth Li. Communicating activations between language model agents. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2025.

Subhro Roy and Dan Roth. Solving general arithmetic word problems. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

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 *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025.

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

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations (ICLR)*, 2021.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of llm agents. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Y Zou. Mixture-of-agents enhances large language model capabilities. In *International Conference on Learning Representations (ICLR)*, 2025.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Haoyi Wu, Zhihao Teng, and Kewei Tu. Parallel continuous chain-of-thought with jacobi iteration. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Xinlei Yu, Zhangquan Chen, Yongbo He, Tianyu Fu, Guanting Dong, Cheng Yang, Chengming Xu, Yue Ma, Xiaobin Hu, Zhe Cao, et al. The latent space: Foundation, evolution, mechanism, ability, and outlook. *arXiv preprint arXiv:2604.02029*, 2026.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. MasRouter: Learning to route LLMs for multi-agent systems. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for LLM-based multi-agent systems. In *International Conference on Learning Representations (ICLR)*, 2025a.

Ruichen Zhang, Mufan Qiu, Zhen Tan, Mohan Zhang, Xiaopeng Lu, Jie Peng, Kaidi Xu, Leandro Z. Agudelo, Peter Zhenghao Qian, and Tianlong Chen. Symbiotic cooperation for web agents: Harnessing complementary strengths of large and small LLMs. In *Conference on Parsimony and Learning*, 2026.

Zhen Zhang, Xuehai He, Weixiang Yan, Ao Shen, Chenyang Zhao, and Xin Wang. Soft thinking: Unlocking the reasoning potential of llms in continuous concept space. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025b.

Yujia Zheng, Zhuokai Zhao, Zijian Li, Yaqi Xie, Mingze Gao, Lizhu Zhang, and Kun Zhang. Thought communication in multiagent collaboration. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Jiaru Zou, Rui Pan, Ruizhong Qiu, Pan Lu, Shizhe Diao, Jindong Jiang, Hanghang Tong, Tong Zhang, Markus J Buehler, Jingrui He, et al. Recursive multi-agent systems. *arXiv preprint arXiv:2604.25917*, 2026a.

Jiaru Zou, Xiyuan Yang, Ruizhong Qiu, Gaotang Li, Katherine Tieu, Pan Lu, Ke Shen, Hanghang Tong, Yejin Choi, Jingrui He, James Zou, Mengdi Wang, and Ling Yang. Latent collaboration in multi-agent systems. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026b.

APPENDIX

A EXPERIMENTS SETUPS

A.1 TASKS AND DATASETS.

Math Benchmarks. Following the dataset choices of Deng et al. (2025), we train INFOLAT on GSM8K-aug (Cobbe et al., 2021), an augmented version of the GSM8K training set that expands the original 7,473 problems to 385,000 samples through GPT-4 prompting. We evaluate on four test sets: GSM8K-aug (1,319 problems), GSM-Hard (Gao et al., 2023) (1,319 problems obtained by replacing the numbers in the GSM8K test set with larger, less common values), MultiArith (Roy & Roth, 2015) (180 problems), and SVAMP (Patel et al., 2021) (300 problems). Each benchmark is a set of arithmetic word problems that require multi-step reasoning, and the answer is a single number.

ALFWorld. Following Interlat (Du et al., 2026), we adopt ALFWorld (Shridhar et al., 2021) as our embodied benchmark, which is a text-only benchmark that simulates embodied household tasks, where an agent observes textual scene descriptions and issues high-level commands from a constrained action set, such as `go to`, `take`, and `put`. Each task is long-horizon and compositional, requiring the agent to perceive, plan, and act over multiple steps under partial observability. We train on the data collected by Song et al. (2024), which contains 3,119 trajectories spanning six task templates: four manipulation templates—pick-and-place, pick-clean-then-place, pick-heat-then-place, and pick-cool-then-place—and two observation templates—look-in-receptacle and look-at-object. We evaluate on the official seen and unseen splits, which contain 140 and 134 tasks, respectively.

A.2 PROMPT TEMPLATE.

Both benchmarks share the same family-specific chat templates. The prompt of the Llama-family receivers is formatted as

```
<|start_header_id|>user<|end_header_id|>
{instruction}

{problem}<|eot_id|><|start_header_id|>assistant<|end_header_id|>
```

and the prompt of the Qwen-family receivers is formatted as

```
<|im_start|>system
{instruction}
<|im_end|>
<|im_start|>user
{problem}<|im_end|>
<|im_start|>assistant
```

where {`instruction`} is replaced by the benchmark-specific instruction and {`problem`} by the task input.

For the math benchmarks, the instruction is "Please reason step by step, and put your final answer within `\boxed{}`.", and {`problem`} is replaced by the actual question.

For ALFWorld, the instruction is:

```
You are an agent in the ALFWorld text environment.
Given the task and current progress below, reason briefly
about the situation, then output the next TextWorld command.
Reply in this exact format:
<think>
<your reasoning>
</think>
\boxed{go to dresser 1}

The available actions are:
1. go to {recep}
2. take {obj} from {recep}
3. put {obj} in/on {recep}
4. open {recep}
5. close {recep}
6. toggle {obj} {recep}
7. use {obj}
8. clean {obj} with {recep}
9. heat {obj} with {recep}
10. cool {obj} with {recep}

Use the object and receptacle names and indices appearing in
the observation. If the environment returns "Nothing happens.",
do not repeat the previous command; choose another valid action.

Think rules:
- Write in first person and ground the reason in the current
  observation and recent history, including what you see,
  what you are holding, whether a place was empty or closed,
  and whether you just found the required object.
- Usually use 2--3 sentences to justify the immediate next
  step toward completing the task.
- Prefer concrete object and receptacle names from the scene.
  If a location contains nothing useful, search elsewhere
  rather than inventing objects.
- Do not write a long multi-step plan; explain only why the
  next command is appropriate for the current state.

Action rules:
- Put exactly one TextWorld command inside \boxed{...},
  matching one of the templates above.
- Do not write anything after the \boxed{...} line.
```

Here, {problem} contains the task description, the current observation, and the complete history of preceding observations and actions. Only the command inside \boxed{...} is executed.

A.3 Training and Implementation Details.

Hardware Settings. All experiments are conducted on 8 NVIDIA A800 40GB GPUs. Training uses bfloat16 mixed precision and DeepSpeed ZeRO-2, and inference uses FlashAttention-2.

Math Benchmarks. Before training, we align each receiver's vocabulary with the sender's, following Section 3.3. The Llama-3.2-1B-Instruct receiver shares the sender's vocabulary and uses its existing input embeddings. In contrast, the Qwen2.5-1.5B-Instruct receiver extends its communication vocabulary with 18,434 tokens that are present in the sender's vocabulary but absent from its own, and constructs their input embeddings. Table 4 summarizes the vocabulary statistics.

Following the embedding construction in Section 3.3, the receiver constructs an input embedding for each missing token by tokenizing its text with τ_R and combining the embeddings of the resulting

Table 4: Vocabulary statistics for latent communication on the math benchmarks.

Quantity	Llama	Qwen
Usable vocabulary	128,000	151,643
Intersection	109,566	
New tokens	0	18,434

tokens. For instance, the Llama tokenizer groups digits into runs, so "48" is a single token in the sender's vocabulary, whereas the Qwen tokenizer emits one digit per token and spells it as "4" and "8". Table 5 lists several representative examples.

Table 5: Examples of constructing receiver input embeddings for sender tokens that are missing from the Qwen receiver's vocabulary.

Llama token	Qwen token sequence
`48`	`4 + 8`
`400`	`4 + 0 + 0`
`365`	`3 + 6 + 5`
`lerini`	`ler + ini`
`␣tarih`	`␣t + arih`

We then train INFOLAT on GSM8K-aug through the three-stage pipeline described in Sections 3.2–3.5: latent label construction, sender training, and receiver training. All trainable components use LoRA with rank 64 and dropout 0.1, AdamW with a 5% warm-up, weight decay 0.01, and a cosine schedule.

Specifically, for latent label construction, we use Llama-3.2-3B-Instruct as the message encoder and Llama-3.2-1B-Instruct as the predictive decoder, and train them with a learning rate of 1e-4 for 10 epochs, using a per-device batch of 4 with 16 gradient-accumulation steps. For sender training, we initialize the sender from Llama-3.2-3B-Instruct and train it with a learning rate of 3e-4 for 30 epochs, using a per-device batch of 16 with 8 gradient-accumulation steps. For receiver training, we initialize each receiver from Llama-3.2-1B-Instruct and Qwen2.5-1.5B-Instruct respectively, and train it with a learning rate of 3e-4 for 1 epoch, using a per-device batch of 64. The latent message uses a top-10 interpolation and ρ=2.

At inference, a single Llama-3.2-3B-Instruct sender serves both receivers: it broadcasts the same latent message to the Llama-3.2-1B-Instruct and Qwen2.5-1.5B-Instruct receivers, and we evaluate the performance of each sender–receiver pair. All methods are evaluated with the same decoding budget: temperature 0.6, top-p 0.95, and at most 256 new tokens, over three random seeds 42, 1234, and 2024, and we report the mean and the sample standard deviation. Checkpoints are selected on a held-out validation split of 500 examples from GSM8K-aug.

ALFWorld. All models are trained on ALFWorld trajectory data containing task descriptions, plans, and actions (Song et al., 2024). We derive training instances from these expert trajectories. The input to each instance consists of the task description, the corresponding expert plan message, and the action, which serve as supervision. We reserve 5% of the episodes for validation and perform the split at the episode level.

Since ALFWorld uses Qwen2.5-7B-Instruct as the sender, the direction of vocabulary alignment is reversed relative to the math benchmarks. The Qwen2.5-1.5B-Instruct receiver shares the sender's vocabulary and reuses its existing input embeddings, requiring no additional communication entries. In contrast, the Llama-3.2-1B-Instruct receiver adds 42,077 sender-side entries that are absent from its native vocabulary and constructs their input embeddings following the embedding construction in Section 3.3. Table 6 summarizes the resulting vocabulary statistics.

All trainable components use LoRA with rank 64 and dropout 0.1. We use AdamW with a 5% warm-up ratio, weight decay of 0.01, and a cosine learning-rate schedule. For latent label construction, Qwen2.5-7B-Instruct serves as the message encoder and Qwen2.5-1.5B-Instruct as the predictive decoder. We first optimize the encoder and decoder separately for 15 and 10 epochs, respectively,

Table 6: Vocabulary statistics for latent communication on the alfworld benchmarks.

Quantity	Llama	Qwen
Usable vocabulary	128,000	151,643
Intersection	109,566	
New tokens	42,077	0

and then jointly optimize both components for 6 epochs. All three optimization steps use a learning rate of 1e-4 and a per-device batch size of 2, with 8, 8, and 16 gradient-accumulation steps.

We train the sender for 50 epochs with a learning rate of 3e-4, a per-device batch size of 2, and 8 gradient-accumulation steps. The trained sender is then frozen, and each receiver is trained independently to predict the next action conditioned on the constructed latent labels. Each receiver is trained for 5 epochs with a learning rate of 3e-4, a per-device batch size of 4, and 1 gradient-accumulation step. Latent messages use top-10 interpolation with a compression ratio of 2.

At inference, the trained sender is paired separately with each receiver. During interaction, the sender produces latent messages conditioned on the task description, which guide the receiver's action prediction. An episode terminates when the task is completed or the maximum horizon of 20 environment steps is reached. We allow at most 64 latent tokens per message and 96 newly generated tokens for receiver decoding. Evaluation is conducted on the official seen and unseen ALFWorld splits, with task success rate as the primary metric.

A.4 Baseline Settings.

Text Communication. In the Text Communication baseline, the sender generates its reasoning as plain text, and the receiver uses this message together with the task input to produce its answer. To support a fair comparison, both the sender and receiver are fine-tuned on task training data before evaluation.

Relevant Latent Communication Methods. We reproduce InterLat and RecursiveMAS following the training protocols in their original papers (Du et al., 2026; Zou et al., 2026a), retaining their prescribed training objectives and stage-wise schedules. This preserves each baseline's intended training procedure and budget. Our results therefore compare performance under the respective training protocols.

Both methods transmit the sender's raw hidden states as the message, so the message lives in the sender's own latent space, whose dimensionality and geometry are specific to the sender backbone. To make these hidden states consumable, each method learns a projection from the sender's hidden space into the receiver's input embedding space: InterLat trains a lightweight communication adapter (a self-attention and projection layer) jointly with the receiver to rescale and interpret the received hidden states, while RecursiveMAS learns an outer RecursiveLink that linearly maps the sender's embeddings into the receiver's embedding space. Since this projection depends on the hidden dimensionality and geometry of both the sender and the receiver, it must be relearned whenever either endpoint is replaced, binding both methods to a specific sender–receiver pair.

Math Benchmarks For the math benchmarks, the two methods are both trained on GSM8K-aug, and they adopt different training pipelines but share the same inference behavior: the sender reads the question and transmits a latent message, and the receiver then generates the task output. Both methods fix the latent message length during training, and they set this length to 15 steps, which is half of the average text message length in the training data, matching ρ=2 in our method.

InterLat requires the sender and each receiver to be trained as a one-to-one pair, so we train two separate pairs, one for the Llama-3.2-1B-Instruct receiver and one for the Qwen2.5-1.5B-Instruct receiver. Each pair adopts a two-stage protocol. In Stage 1, the receiver and an input projector are trained jointly on the sender's cached hidden states, supervising the receiver's task output, for 10 epochs with batch 8 $\times$ accumulation 4 and learning rate 5e-5. The projector maps the 3072-dimensional sender hidden states into the receiver's embedding dimension, which is 1536 for the Qwen2.5-1.5B-Instruct receiver and 2048 for the Llama-3.2-1B-Instruct receiver. In Stage 2, the

receiver and the projector are frozen, and the sender is trained to generate a latent message of $K{=}15$ steps from the question alone, for 3 epochs with batch 1 $\times$ accumulation 32 and learning rate 5e-5.

RECURSIVEMAS trains only lightweight link modules and keeps all backbones frozen. It adopts a two-stage protocol: the sender's InnerLink is first optimized on the frozen Llama-3.2-3B-Instruct backbone, and a pair-specific OuterLink is then trained for each sender–receiver pair to map the latent message into the receiver's embedding space. Each stage runs 3 epochs with an effective batch of 100 and learning rate 5e-4, with the frozen backbones kept in bfloat16 and the link parameters in FP32. At inference, the sender autoregressively rolls out a latent message of 15 steps.

ALFWorld Both baselines are trained on the same ALFWorld data source as INFOLAT. Following the official settings and architecture design of the original papers, both methods adopt the same interaction protocol: the sender transmits a plan as a latent message, which the receiver then executes. Both methods fix the latent message length during training and set this length to $K{=}64$ steps, which is half of the average text message length in the training data, matching $\rho{=}2$ in our method.

INTERLAT requires the sender and each receiver to be trained as a one-to-one pair, so we train two separate pairs with Qwen2.5-7B-Instruct as the sender, one for the Llama-3.2-1B-Instruct receiver and one for the Qwen2.5-1.5B-Instruct receiver. Each pair adopts a two-stage protocol. In Stage 1, the receiver and an input projector are trained jointly on the sender's cached hidden states of the full-length plan, supervising the receiver's actions with the cross-entropy, conditional-separation, and plan-alignment objectives from the original paper, using AdamW with a global batch size of 16, a learning rate of 1e-5, and a 3% linear warmup. The projector maps the 3584-dimensional sender hidden states into the receiver's embedding dimension, which is 1536 for the Qwen2.5-1.5B-Instruct receiver and 2048 for the Llama-3.2-1B-Instruct receiver. In Stage 2, the receiver and the projector are frozen, and the sender is trained to generate a compressed latent message of $K{=}64$ steps from the task description, using a learning rate of 5e-5 with unit weights on the three compression objectives; the best checkpoint is selected on a 5% validation split.

RECURSIVEMAS trains only lightweight link modules and keeps all backbones frozen. It adopts a two-stage protocol: the sender InnerLink is first optimized on the frozen Qwen2.5-7B-Instruct backbone, and a pair-specific OuterLink is then trained for each sender–receiver pair to map the sender's autoregressively generated latent plan into the receiver's embedding space. Both stages run for 20,000 optimization steps with a batch size of 1 and a learning rate of 5e-4. The OuterLink is trained with action-level cross-entropy to align sender-generated latent plans with the receiver's action predictions. At inference, the sender autoregressively generates a compressed latent message of $K{=}64$ tokens from the task description.

Shuffled Latent Message. The SHUFFLED baseline reuses the same sender–receiver pair as INFOLAT and keeps the receiver's question unchanged, but replaces each receiver's latent message with one generated from a different task. Specifically, the sender first generates a latent message for every test question, and these messages are then reassigned across the questions according to a derangement, so that no example ever receives the message generated for its own question. The pairing is deterministic given a fixed random seed, which makes the shuffled condition reproducible. Since the receiver still answers its own question but conditions on a message about a different task, a performance drop relative to INFOLAT indicates that INFOLAT indeed exploits the task information encoded in the latent message.

A.5 MESSAGE LENGTH AND COMMUNICATION COST

Communication involves three stages: message generation, transmission, and reception. Here, communication cost includes the inference computation required to produce and consume a message, in addition to its transmission.

Generation. Both text and latent messages require one sender Transformer forward pass per autoregressive step. INFOLAT selects the top-K candidates from a single vocabulary-score vector and forms a weighted sum of their embeddings; $K = 10$ does not require ten Transformer forward passes. The embedding construction adds little computation relative to the backbone forward pass, while a shorter generated sequence requires fewer decoding steps. The sender backbone is the same for TEXT COMMUNICATION and INFOLAT in each comparison, including Qwen2.5-7B-Instruct

on ALFWorld. Thus, using a 7B sender does not remove the computational savings from fewer generation steps: both methods incur the forward-pass cost of that same backbone.

Transmission. A latent token carries K vocabulary IDs and weights, so it has a larger payload than a text token. Consequently, a reduction in $\#L$ is not a claim of an equal reduction in transmitted bytes. For short messages in deployments where LLM computation dominates, however, transferring either payload contributes negligibly compared with model prefill and decoding. In this regime, generation and reception determine the dominant communication cost.

Reception. The receiver converts each latent token into one embedding before prefill. From this point onward, latent and text embeddings occupy the same kind of input position and undergo the same Transformer computation. Fewer message positions therefore reduce message-related prefill work, KV-cache storage, and attention work during subsequent decoding.

Thus, $\#L$ effectively reflects communication cost by capturing the computation required to generate and process messages.

B Method Details

B.1 Vocabulary Alignment Calibration

For each sender token $v_i \in \mathcal{V}_S$ that is missing from the receiver's vocabulary, we first compose an uncalibrated embedding $\tilde{\mathbf{e}}_R(v_i)$ following the embedding construction in Section 3.3. To account for the scale and mean direction of native receiver embeddings, we calibrate the combined vector as

$$\mathbf{e}_R(v_i) = a_R \boldsymbol{\mu}_R + b_R \frac{\tilde{\mathbf{e}}_R(v_i) - (\tilde{\mathbf{e}}_R(v_i)^\top \boldsymbol{\mu}_R)\boldsymbol{\mu}_R}{\|\tilde{\mathbf{e}}_R(v_i) - (\tilde{\mathbf{e}}_R(v_i)^\top \boldsymbol{\mu}_R)\boldsymbol{\mu}_R\|_2}, \tag{4}$$

where $\boldsymbol{\mu}_R$ is the estimated unit mean direction of native receiver embeddings; a_R and b_R are their mean projection onto $\boldsymbol{\mu}_R$ and mean orthogonal component norm, respectively.

C Ablation Settings

The ablations in Table 3 use Llama-3.2-3B-Instruct as the sender and Qwen2.5-1.5B-Instruct as the receiver on the math benchmarks.

Latent Label Construction. These variants modify the three-step optimization schedule for the message encoder and predictive decoder within the latent label construction stage (Section 3.2). The full procedure first trains the message encoder with the predictive decoder frozen, then trains the predictive decoder with the message encoder frozen, and finally jointly fine-tunes both components. *w/o decoder-only* skips the decoder-only step and jointly fine-tunes both components immediat ely after training the message encoder. *encoder only* retains only the encoder-only step, with supervision provided through the frozen predictive decoder. *joint only* skips the encoder-only and decoder-only steps and jointly trains the message encoder and predictive decoder from their initial checkpoints. All retained optimization steps use the token-level cross-entropy loss $\mathcal{L}_{\text{label}}$ in equation 1 to supervise subsequent message prediction and task output prediction. After training, the message encoder constructs latent labels M^{label} from the task input x and reference text message M^{text} for subsequent sender and receiver training.

Vocabulary Alignment. *w/o alignment* preserves the sender's full communication vocabulary but omits the receiver-side vocabulary extension and embedding construction in Section 3.3. For each non-special sender token, we search the receiver's vocabulary for a token with the same text. If a match exists, the receiver uses its existing input embedding; otherwise, the receiver input embedding is set to the zero vector, $\mathbf{e}_R(v_i) = \mathbf{0}$. Unmatched tokens therefore contribute zero to the weighted sum used to construct a receiver latent token embedding.

Intersection uses the same token-matching criterion as Section 3.3: for each non-special token in the sender's vocabulary, we search the receiver's vocabulary for a token with the same text. Only

matched tokens are retained for latent communication; sender tokens without a match are excluded instead of being added to the receiver's vocabulary. The matching is based on token text, not token IDs, and each model uses its own existing input embedding for a matched token. For example, if the sender has a token for "48" but the receiver represents it with two tokens, "4" and "8", then "48" is excluded from the communication vocabulary. The sender's latent token distributions are restricted to the retained tokens, while both models keep their original vocabularies for regular text processing and generation.

When a single communication vocabulary is shared by multiple heterogeneous models, each retained token must have a matching token with the same text in every model's vocabulary. If an added model lacks any of these tokens, the intersection shrinks, requiring the communication vocabulary to be rebuilt and the affected models to be retrained. Our vocabulary alignment preserves the sender's communication vocabulary: each receiver reuses its existing input embeddings for matched tokens and constructs input embeddings for missing tokens. Adding a receiver therefore does not change the sender's communication vocabulary or require retraining the sender, although the new receiver still requires receiver training.

Latent Message Representation. *Hidden State* represents each latent token using the sender's last-layer hidden state $\mathbf{h}_t^S \in \mathbb{R}^{d_S}$ instead of vocabulary tokens and their continuous weights. A trainable communication module $P_{S\to R} : \mathbb{R}^{d_S} \to \mathbb{R}^{d_R}$ maps each hidden state into the receiver's input embedding space. For the Llama-3.2-3B-Instruct sender and Qwen2.5-1.5B-Instruct receiver, $d_S = 3072$ and $d_R = 1536$. The sender states are extracted after the final normalization of the residual stream and have a measured root mean square (RMS) of approximately 1.47, whereas the receiver's native input embeddings have an RMS of approximately 0.026. We adapt the two-layer MLP architecture in CODI (Shen et al., 2025) for this projector, using GELU between the two linear layers and LayerNorm after the second layer. To account for this scale difference, we initialize the LayerNorm scale parameter to the RMS of the receiver's input embedding matrix, approximately 0.026, giving the projected states a comparable initial magnitude to native input embeddings. The projector contains approximately 7.08M parameters in this configuration.

The sender generates latent messages autoregressively by feeding each last-layer hidden state back as the next input embedding, following Coconut (Hao et al., 2025). Before feeding it back, we rescale the hidden state to match the RMS of the sender's input embedding matrix. We replace the sender's KL loss in equation 2 with a scale-normalized smooth-L_1 regression loss against teacher hidden states, inspired by the hidden-state distillation in CODI (Shen et al., 2025).

After sender training, we freeze the sender and jointly train a newly initialized projector and the receiver's LoRA parameters with a cross-entropy loss on the task output y. The projector is specific to the sender–receiver pair, so adding a receiver requires training a new projector and receiver LoRA parameters.

Receiver Training. *w/o receiver training* omits the receiver optimization in Section 3.5 and uses the receiver's original Instruct checkpoint with the trained sender and the vocabulary-aligned communication interface. This tests whether vocabulary-compatible latent inputs can be used without receiver adaptation.

D Analysis Details

D.1 Diagnosing Message Collapse via Shuffling

To verify the message-collapse observation in the Introduction, we compare each method's task performance with original question–message pairs against performance when the messages are shuffled across problems, keeping the receiver's problem input unchanged. Figure 6 reports this comparison on the four math benchmarks, with Llama-3.2-3B-Instruct as the sender and Llama-3.2-1B-Instruct (left column) and Qwen2.5-1.5B-Instruct (right column) as the receivers. InterLat and RecursiveMAS achieve nearly unchanged performance under shuffling, indicating that their latent messages collapse into task-agnostic messages. In contrast, Text Communication and InfoLat both drop substantially, showing that their messages convey task-specific information that the receiver relies on.

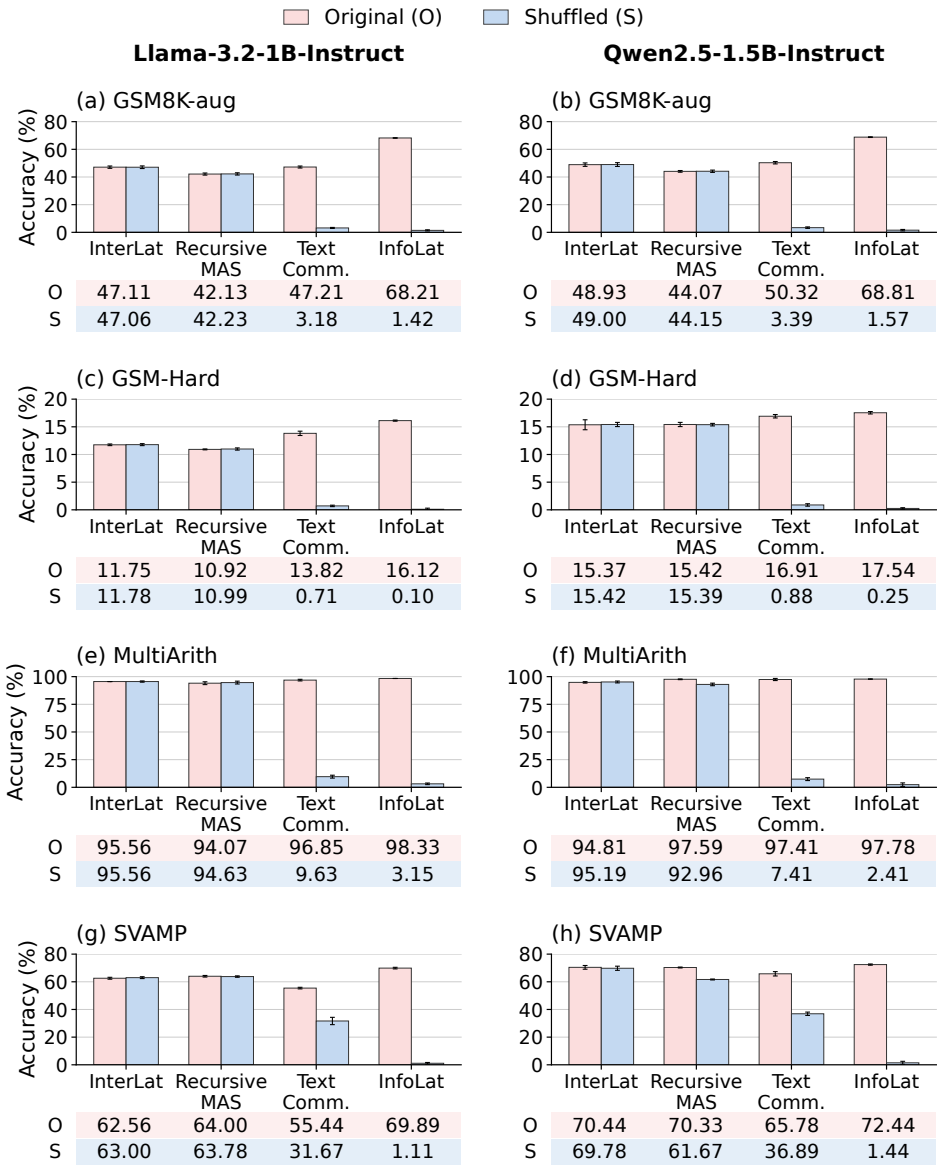
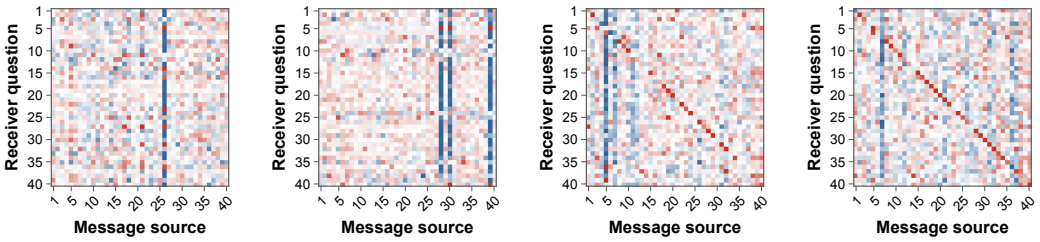


Figure 6: Original vs. shuffled task performance, with Llama-3.2-3B-Instruct as the sender. Columns correspond to receivers and rows to datasets. Original messages come from the same problem; shuffled messages come from a different problem. Values below each panel report mean accuracy for original (O) and shuffled (S) messages; error bars show standard deviations. Text Comm. denotes TEXT COMMUNICATION and Ours denotes INFOLAT.

D.2 QUESTION–MESSAGE COMPATIBILITY

Experimental Setup. For the heatmaps in Figures 4 and 7, we randomly sample $n_{\mathrm{q}} = 40$ problems from the GSM8K-aug test set and evaluate all n_{q}^2 question–message combinations for each method. We use Llama-3.2-3B-Instruct as the sender and report results for both the Qwen2.5-1.5B-Instruct receiver (Figure 4) and the Llama-3.2-1B-Instruct receiver (Figure 7), comparing INTERLAT, RECURSIVEMAS, TEXT COMMUNICATION, and INFOLAT. Following the notation in Section 3.1, let $x^{(i)}$ denote the i-th task input and $M^{(j)}$ the message generated from $x^{(j)}$. Parenthesized superscripts index examples, while token positions are indexed by subscripts. Here, each task input is a math question, and $y^{(i)}$ is the corresponding ground-truth task output, tokenized by the

receiver, with $L_y^{(i)}$ output tokens. For each row i, we keep the receiver's task input $x^{(i)}$ and task output $y^{(i)}$ unchanged and replace the message with $M^{(j)}$ from each sampled question. Thus, diagonal entries correspond to matched question–message pairs, while off-diagonal entries correspond to mismatched pairs.



(a) InterLat (b) RecursiveMAS (c) Text Communication (d) INFOLAT

Figure 7: Question–message compatibility on 40 randomly sampled GSM8K-aug test problems, with Llama-3.2-3B-Instruct as the sender and Llama-3.2-1B-Instruct as the receiver. Rows denote receiver questions and columns denote message sources. Red indicates higher compatibility; diagonal entries correspond to matched question–message pairs.

Compatibility Score. We condition the receiver on the task input $x^{(i)}$ and message $M^{(j)}$, and compute the average token-level log-probability of the task output $y^{(i)}$ under teacher forcing:

$$s_{ij} = \frac{1}{L_y^{(i)}} \sum_{t=1}^{L_y^{(i)}} \log p_\phi(y_t^{(i)} \mid x^{(i)}, M^{(j)}, y_{<t}^{(i)}). \tag{5}$$

At each step, the receiver is given the preceding ground-truth output tokens $y_{<t}^{(i)}$ to predict the next token $y_t^{(i)}$. Since some questions are easier than others, their task outputs may receive higher log-probabilities regardless of the received message. We remove differences in row means and scales by defining the *question–message compatibility score* as

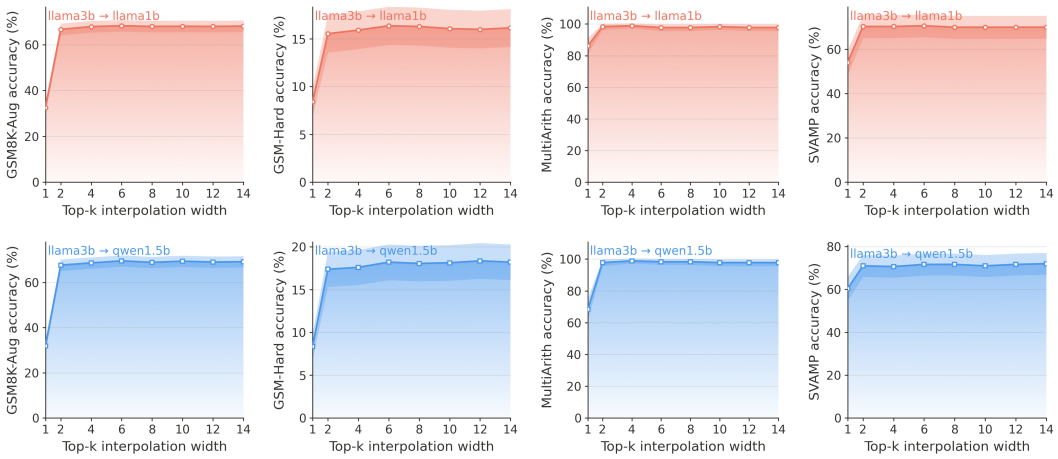
$$C_{ij} = \frac{s_{ij} - \bar{s}_i}{\sigma_i}, \qquad \bar{s}_i = \frac{1}{n_{\mathrm{q}}} \sum_{j=1}^{n_{\mathrm{q}}} s_{ij}, \qquad \sigma_i^2 = \frac{1}{n_{\mathrm{q}}} \sum_{j=1}^{n_{\mathrm{q}}} (s_{ij} - \bar{s}_i)^2. \tag{6}$$

The row mean $\bar{s}_i$ and standard deviation σ_i are computed over all message sources within each row, including the matched message, using the population standard deviation (`ddof=0`). Normalization is performed independently for each method.

A positive C_{ij} means that, given message $M^{(j)}$, the receiver assigns a higher log-probability to the task output $y^{(i)}$ than it does on average over the sampled messages. A clear positive diagonal indicates that messages generated from the same question generally receive higher scores than messages from other questions. This provides evidence that the messages carry task-specific information that the receiver uses for task output generation. The absence of a diagonal pattern indicates that messages from the same question offer little systematic advantage under this measure. The normalized scores measure this relative advantage; they do not measure the absolute change in log-probability when a message is replaced.

D.3 TOP-K SENSITIVITY

We examine the sensitivity of INFOLAT to the interpolation width K, which controls the number of vocabulary tokens used to represent each latent token, over $K \in \{1, 2, 4, 6, 8, 10, 12, 14\}$. Figure 8 reports accuracy on four math benchmarks with Llama-3.2-3B-Instruct as the sender and either Llama-3.2-1B-Instruct or Qwen2.5-1.5B-Instruct as the receiver. Across both receivers and all four benchmarks, increasing K from 1 to 2 substantially improves accuracy, while further increases yield comparatively small changes. These results suggest that combining multiple vocabulary tokens is beneficial and that performance is relatively insensitive to the precise interpolation width once $K \geq 2$, consistent with our default choice of $K = 10$.

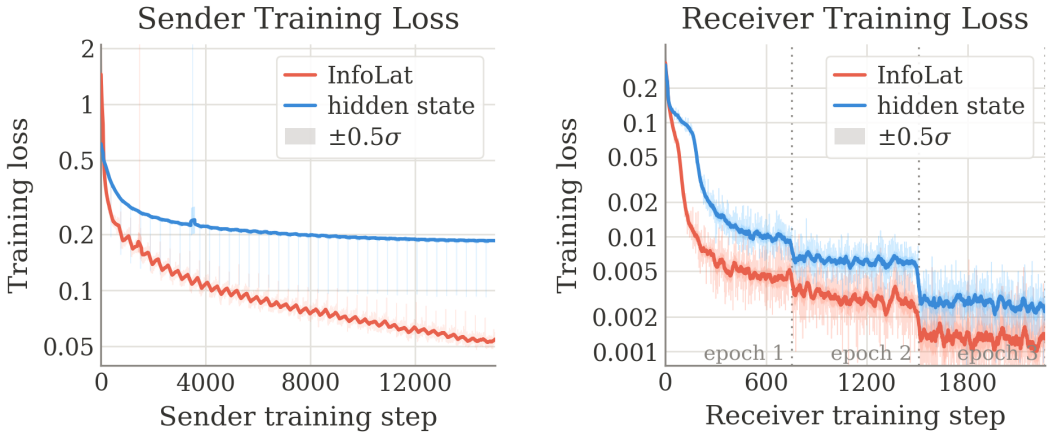


(a) GSM8K-aug **(b) GSM-Hard** **(c) MultiArith** **(d) SVAMP**

Figure 8: Sensitivity to the top-K interpolation width. The sender is Llama-3.2-3B-Instruct; the top and bottom rows use Llama-3.2-1B-Instruct and Qwen2.5-1.5B-Instruct as receivers, respectively. Columns show GSM8K-aug, GSM-Hard, MultiArith, and SVAMP, from left to right.

D.4 Latent Message Representation

We compare INFOLAT with the *Hidden State* variant described in Appendix C, using Llama-3.2-3B-Instruct as the sender and Qwen2.5-1.5B-Instruct as the receiver. INFOLAT represents each latent token as a set of vocabulary tokens and their continuous weights, while the hidden-state variant uses the sender's last-layer hidden state and a learned projector to construct the receiver's input embeddings. Figures 5 and 9 compare their task accuracy and training losses, respectively.



(a) Sender training (b) Receiver training

Figure 9: Training losses for INFOLAT and the hidden-state variant, with Llama-3.2-3B-Instruct as the sender and Qwen2.5-1.5B-Instruct as the receiver. Sender losses use different objectives and are not directly comparable in magnitude; both receiver losses use cross-entropy on the task output. Loss axes are logarithmic.

Sender Training. The sender loss for INFOLAT continues to decrease throughout the displayed training steps, while the hidden-state variant shows progressively smaller reductions after its initial decrease (Figure 9(a)). The two variants use different sender objectives: INFOLAT minimizes the KL divergence in equation 2, whereas the hidden-state variant minimizes a scale-normalized smooth-L_1 regression loss. Their absolute sender loss values are therefore not directly comparable; these curves illustrate the optimization trajectory of each objective.

Receiver Training and Task Performance. Both receivers are trained with a cross-entropy loss on the task output y. As shown in Figure 9(b), INFOLAT reduces this loss more quickly and maintains a lower loss over the displayed training steps. This training advantage is accompanied by higher task accuracy in Figure 5: INFOLAT reaches 69.5% at receiver training step 1,425, while the hidden-state variant reaches its highest displayed accuracy of 44.8% at step 225 and remains below this value with further training. The continued decrease in the hidden-state variant's receiver loss does not translate into a corresponding improvement in task accuracy.

Although the *Hidden State* variant can reuse a frozen sender through receiver-specific projectors, fixing the output dimension only ensures interface compatibility, not effective use of the messages. Its lower accuracy and larger fluctuations across receiver training checkpoints suggest that, in the evaluated setting, this hidden-state interface is less suitable than INFOLAT's latent messages for effective sender reuse across heterogeneous receivers.