LATENT BRIDGE: ASKING FOR HELP WITHOUT WORDS

Anonymous authors
Paper under double-blind review

ABSTRACT

Text-based model collaboration requires a large model to generate intermediate responses before a smaller model can use its guidance, adding large-model decoding cost to inference. We introduce POND (**P**refill-**O**nly, **N**o-**D**ecode), a method that uses a frozen large sender for prefill and a small receiver for all autoregressive decoding. A trainable bridge maps the sender's hidden states to additional attention keys and values, allowing the receiver to use these representations throughout generation. POND also supports periodic review, in which the sender processes the receiver's partial response and updates its guidance through additional prefill passes while preserving the receiver's existing KV cache. Across reasoning and coding benchmarks, POND improves receiver performance with senders from different model families. In particular, GLM-5.3 provides gains comparable to those of Qwen3.8-Max when paired with a Qwen receiver, supporting the effectiveness of representation transfer between independently trained model families. POND further demonstrates its ability to generalize across domains. When trained on math and coding tasks, it transfers directly to general knowledge tasks without additional domain-specific training. These results demonstrate that large frontier model prefill representations can improve small-model generation without requiring large-model autoregressive decoding. Code and inference examples can be found at `https://latent-1117.pages.dev`.

1 INTRODUCTION

Natural language is the primary means of human communication, and multi-model systems often mimic this approach by exchanging generated text. One model generates an intermediate response that another model then uses as input (Du et al., 2023; Wu et al., 2023; Wang et al.). Text supports communication between models with different architectures, but requires the sender to generate an intermediate response and then the receiver to process it. Such communication can be particularly costly when a frontier-scale model assists a smaller model deployed on edge hardware. In this setting, the large model decodes the intermediate response that will be sent the small model token by token, so the system incurs large-model decoding latency even though the small model produces the final answer. These limitations motivate a key question:

Can a large model improve a small model's generation without incurring the latency of autoregressive decoding?

Our approach rests on an asymmetry in large-model inference cost. During prefill, a model processes all input tokens in parallel; the computation is compute-bound and uses the hardware efficiently. Standard autoregressive decoding instead requires one forward pass per generated token. These passes must run sequentially, so total decoding latency grows with response length, making long responses particularly time-consuming to generate (Patel et al., 2024; Zhong et al., 2024). Prefill produces contextual representations of the input without requiring the large model to generate text. We therefore investigate how much of the benefit of large-model assistance can be recovered by transferring these representations directly to a small model.

We refer to the large model as the *sender* and the small model as the *receiver*. The main challenge is that the two models are trained independently. Their hidden dimensions differ, and even at matched dimensions their representation spaces are not aligned, so the receiver cannot use the sender's states

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.

directly. We address this with a trainable bridge that maps sender hidden states to additional attention keys and values in the receiver. Both models read the same input, and the receiver attends to the bridged keys and values alongside its own context. The bridge is trained with the receiver's next-token prediction loss on target answers, while the sender remains frozen.

We introduce POND (Prefill-Only, No-Decode), a framework for collaboration between a frontier-scale sender and a small receiver capable of running on suitable edge hardware. The sender performs only prefill, while the receiver handles all autoregressive decoding. Throughout generation, the receiver attends to keys and values derived from the sender's representations while retaining its original prompt and KV cache. This design allows the receiver to leverage frontier-model representations without updating the sender's parameters or requiring it to generate intermediate tokens.

A related approach is multi-token prediction (MTP), which trains additional prediction heads or modules to predict multiple future tokens from a model's hidden states (Gloeckle et al., 2024; Liu et al., 2024). These modules can also be used for speculative decoding, where the main model verifies proposed tokens. POND instead transfers hidden states between separately pretrained models and trains the bridge using the receiver's next-token prediction objective. The receiver generates the final response directly, without sender verification or a requirement to preserve the sender's output distribution. The goal is to improve small-model generation using large-model prefill, rather than to accelerate sampling from the large model.

We also extend POND with *periodic review*. In the initial-only setting, the sender processes the prompt once. With review, the sender periodically processes the available response prefix through an additional prefill pass. The bridge maps updated sender representations to keys and values, while the receiver continues decoding from its existing KV cache. Review allows sender guidance to depend on the partial response while keeping all autoregressive decoding in the receiver.

We evaluate POND on math, code, and general-knowledge tasks using three frontier-scale senders and an edge-scale receiver. Initial transfer alone improves receiver performance. Periodic review provides further gains, particularly for tasks requiring long-form generation. By updating the sender representations based on the partial response, periodic review maintains effective guidance over extended decoding. These results show that frontier-model prefill representations can effectively guide small-model generation without requiring large-model autoregressive decoding. Periodically updating these representations further improves their effectiveness for long outputs.

Our contributions are:

- We propose POND, a collaboration framework in which a frozen large sender performs only prefill and a small receiver performs all autoregressive decoding. A trainable bridge maps sender hidden states to additional attention keys and values in the receiver, allowing the two independently trained models to communicate without intermediate text.
- We introduce periodic review, which lets the sender re-read the partial response through additional prefill passes. This allows guidance to adapt to the subsequent response without requiring sender-side decoding or recomputation of the receiver's generated prefix.
- We show that, across frontier-scale senders, POND consistently improves receiver performance on math and code tasks and transfers directly to general knowledge tasks without additional training. These results highlight POND's cross-domain generalizability.

2 Related Work

2.1 Model Communication and Collaboration

Many multi-model systems improve performance by allowing language models to exchange natural-language messages. Prior work has explored communication both within a single model (Qu et al., 2024; Sun et al., 2026b; DeepSeek-AI, 2026) and across multiple models. Multi-agent debate, for example, lets several models propose, critique, and revise their solutions over multiple rounds (Du et al., 2023; Chen et al., 2024). General agent frameworks organize models into specialized roles and coordinate their work through conversation (Wu et al., 2023; Qian et al., 2024). Mixture-of-Agents follows a layered design in which each model conditions on the textual outputs produced by

models in the previous layer (Wang et al.). Other work studies how the communication topology can be made sparse to reduce the token and inference cost of multi-agent discussion (Li et al., 2024).

Text offers an interpretable interface across heterogeneous models, but incurs sender-side decoding costs and limits communication to what the sender expresses in text. Beyond text-based communication, Cache-to-Cache (Fu et al., 2026) enables direct semantic transfer by projecting and fusing KV caches across models. POND instead learns a latent channel that transforms frontier-model hidden states into a separate guidance memory, preserving the receiver's text KV cache. This interface supports both initial transfer and response-dependent periodic review, allowing the sender to provide guidance entirely through prefill while the receiver handles all autoregressive decoding.

2.2 Knowledge Distillation and Latent Reasoning

Knowledge distillation trains a smaller student to reproduce information provided by a larger teacher, commonly through output distributions or teacher-generated sequences (Hinton et al., 2015; Kim & Rush, 2016). Feature-based methods additionally match hidden states, attention patterns, or other intermediate representations (Jiao et al., 2020; Wang et al., 2020). For generative language models, recent methods distill teacher distributions, supervise the student on its own generated sequences, or use teacher-generated rationales as additional training targets (Gu et al., 2024; Agarwal et al., 2024; Hsieh et al., 2023). These methods typically transfer teacher knowledge into a standalone student that operates without the teacher at inference time.

POND instead trains a latent bridge and selected receiver components to use guidance from a frozen sender at inference time. This guidance depends on the current prompt and, under periodic review, on the receiver's partial response. Rather than aiming to produce a standalone student, POND transfers instance-specific computation through prefill representations, without requiring sender-side autoregressive decoding.

A related line of work explores intermediate computation beyond the final visible output. Pause tokens and internal rationales enable additional computation before prediction (Goyal et al., 2024; Zelikman et al., 2024), while latent-reasoning methods use hidden-state feedback or compressed chain-of-thought representations (Hao et al., 2024; Shen et al., 2025). These methods primarily study reasoning within a single model, whereas POND transfers prefill representations between independently trained models without sender-side autoregressive decoding.

3 Method

POND connects a large sender model to a small receiver through a learned latent bridge. The sender only reads, through prefill passes; the receiver generates every output token. A lightweight bridge turns the sender's hidden states into a latent message that the receiver reads as additional KV entries.

3.1 Problem Setup

Let x be the input and $y = (y_1, \ldots, y_N)$ the output. We denote the frozen sender by S_{θ_s}, the receiver by R_{θ_r}, and the bridge parameters by ϕ. In the receiver's token space, x occupies positions $1, \ldots, n_x$ and y_t occupies position $n_x + t$. The receiver generates

$$p_{\theta_r,\phi}(y \mid x) = \prod_{t=1}^{N} p_{\theta_r,\phi}\big(y_t \mid x,\, y_{<t},\, \mathcal{M}_t\big), \tag{1}$$

where $\mathcal{M}_t$ is the latent memory available at step t: a single message computed from x in the basic setting, enlarged during generation by periodic review.

3.2 Latent Compression

We take the final-layer hidden states of the sender, $H = S_{\theta_s}(x) \in \mathbb{R}^{T \times d_S}$, where T is the sender input length and h_j denotes the j-th row. The bridge maps each state to a low-dimensional latent,

$$z_j = \mathrm{MLP}_\phi(h_j) \in \mathbb{R}^{d_z}, \qquad d_z \ll d_S, \tag{2}$$

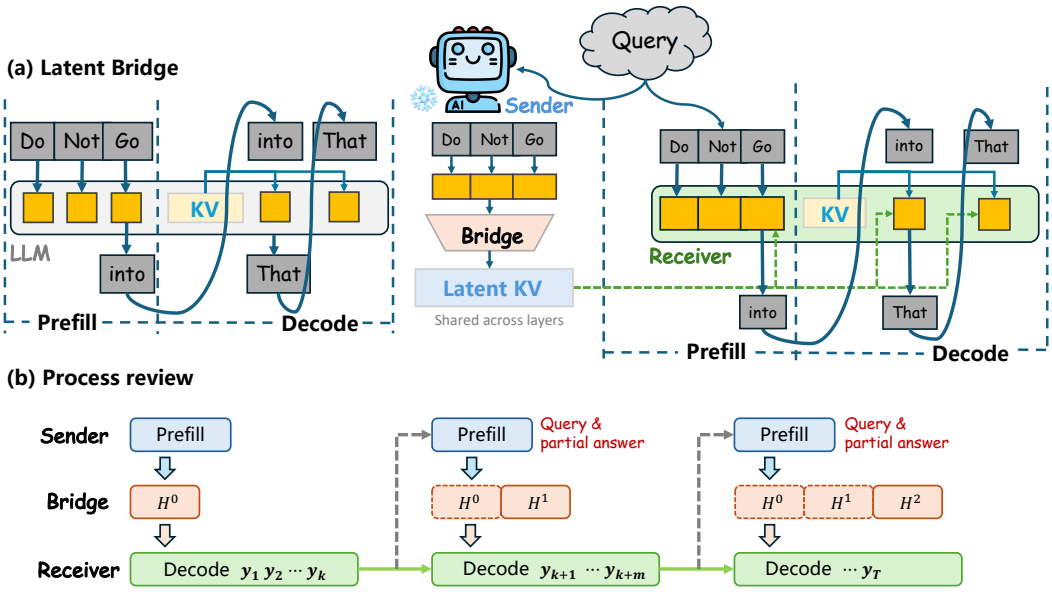

Figure 1: Overview of POND. A frozen sender prefills the input, and a trainable bridge compresses its last layer hidden states into a shared latent message, which layer-specific adapters project into KV entries for the receiver. The receiver, fine-tuned jointly with the bridge, generates the full response. At review boundaries, the sender prefills the newly generated segment and the resulting entries are appended to the latent memory, leaving the receiver's KV cache untouched.

giving the message $Z = B_\phi(x) = [z_1; \ldots; z_T]$. Compression acts on the feature dimension only: each sender token keeps one latent, so every entry has a well-defined location in the text, and a single Z is shared by all receiver layers. Because the sender is causal and B_ϕ acts position-wise, each latent depends only on its own prefix,

$$z_j = f_\phi(x_{\le j}), \tag{3}$$

a property we call *prefix determinism*. It enables both incremental review and single-pass training.

3.3 Layer-Specific KV Adapters

We attach an adapter to each full-attention layer of the receiver. For an adapted layer ℓ, a module $\mathcal{P}_{\phi_\ell}$ (K/V projections followed by RMS normalization) maps the shared latent to keys and values in the receiver's KV-head layout,

$$(\bar{K}^z_\ell, V^z_\ell) = \mathcal{P}_{\phi_\ell}(Z) \in \mathbb{R}^{T \times n_{\mathrm{kv}} \times d_h}, \qquad K^z_{\ell,j} = \mathrm{RoPE}_\ell(\bar{K}^z_{\ell,j}, p_j), \tag{4}$$

where p_j is the receiver position aligned with latent j. The entries are computed once per message and cached. Sharing Z keeps the message compact, while per-layer adapters let each layer read it in its own subspace.

3.4 Gated Joint Attention

At an adapted layer ℓ, let Q_ℓ be the receiver queries and $K^{\mathrm{txt}}_\ell, V^{\mathrm{txt}}_\ell$ the keys and values of the input and the available output prefix, under the usual causal mask. Alongside the original output $A^{\mathrm{txt}}_\ell = \mathrm{Attn}(Q_\ell, K^{\mathrm{txt}}_\ell, V^{\mathrm{txt}}_\ell)$, the receiver computes a joint output normalized over the visible latent entries and the text,

$$A^{\mathrm{joint}}_\ell = \mathrm{Attn}\big(Q_\ell, [K^z_\ell; K^{\mathrm{txt}}_\ell], [V^z_\ell; V^{\mathrm{txt}}_\ell]\big), \tag{5}$$

and combines the two through a learned gate g_ℓ initialized to zero:

$$A_\ell = A^{\mathrm{txt}}_\ell + \tanh(g_\ell)\big(A^{\mathrm{joint}}_\ell - A^{\mathrm{txt}}_\ell\big). \tag{6}$$

A residual view. Let A_ℓ^z be attention over visible latent entries alone, and λ_ℓ^z, $\lambda_\ell^{\text{txt}}$ the log-sum-exp of the latent and text logits. The joint softmax factorizes with $\alpha_\ell = \sigma(\lambda_\ell^z - \lambda_\ell^{\text{txt}})$, so that

$$A_\ell = A_\ell^{\text{txt}} + \tanh(g_\ell)\, \alpha_\ell \big(A_\ell^z - A_\ell^{\text{txt}}\big). \tag{7}$$

The latent branch is thus a residual correction scaled by a static gate and a content-dependent coefficient α_ℓ, and it vanishes whenever no latent entry is visible. The identity also gives an efficient implementation: text attention runs through the receiver's unmodified kernel, and a small latent attention is merged through log-sum-exp statistics, as in split-KV attention.

3.5 Periodic Review

Let $\mathcal{B} = \{b_r \mid r = 0, \ldots, R,\ 0 = b_0 < b_1 < \cdots < b_R < N\}$ denote the review boundaries. At each $b_r \in \mathcal{B}$, the sender reads the partial response $c_r = x \oplus y_{1:b_r}$. By prefix determinism, only the new segment $y_{b_{r-1}+1:b_r}$ produces new latents. The review message is $Z^{(r)} = [z_j]_{j \in J_r}$, where J_r indexes the segment's sender tokens ($Z^{(0)} = B_\phi(x)$). The sender computes it by prefilling only this segment using its cached prefix. The projected entries are appended to the latent memory,

$$\mathcal{M}_\ell^{(r)} = \mathcal{M}_\ell^{(r-1)} \oplus \big(K_\ell^{z,(r)}, V_\ell^{z,(r)}\big), \tag{8}$$

while the receiver's KV cache is left untouched. Since each token passes through the sender once, total sender computation equals a single prefill over $x \oplus y_{1:b_R}$ regardless of the number of reviews; review frequency trades freshness of guidance against the number of sender calls and memory size.

3.6 Causal Visibility and Position Alignment

Position alignment. With a shared tokenizer, p_j is the receiver position of token j. Otherwise we align by character offsets: with e_j and E_q the end offsets of sender token j and receiver token q,

$$p_j = \min\{\, q : E_q \geq e_j \,\}, \tag{9}$$

the first receiver position that has read all text encoded by z_j (details in Appendix B).

Visibility mask. Review r is issued after y_{b_r} is sampled but before the receiver processes it, so its release position is $\rho_r = n_x + b_r$, with $\rho_0 = 0$. A query at position q attends to entry j of message r if and only if

$$\tau_j = \max(p_j, \rho_r) \leq q. \tag{10}$$

Prompt positions thus see the initial message causally by position, response positions see all of it, and review messages become visible upon release. Every visible entry encodes only text already read at q, and the mask reproduces exactly the entries available at inference, so parallel training on full sequences matches incremental deployment.

3.7 Sparse Latent Memory

As the memory grows with every reviewed token, we read it with sparse attention, following Compressed Sparse Attention (DeepSeek-AI, 2026), which builds on DeepSeek Sparse Attention (DeepSeek-AI, 2025); the receiver's text attention is unchanged. Latent entries are compressed along the sequence at ratio m, and a lightweight indexer keeps the top k visible entries per query,

$$\mathcal{I}_\ell(q) = \mathrm{TopK}_k\big\{\, s_\ell(q, j) \,:\, \tau_j \leq q \,\big\}, \tag{11}$$

over which A_ℓ^z and λ_ℓ^z are computed. Compression blocks do not span messages, a compressed entry takes the largest threshold among its constituents, and masking precedes selection, so the guarantees of Section 3.6 are preserved. The latent branch then costs $\mathcal{O}(k)$ attention per query regardless of the number of reviews; m, k, and the indexer are specified in Appendix C.

3.8 Training

With the sender frozen, we train the bridge ϕ (compressor, adapters, gates, and indexer) and all receiver parameters θ_r by full-parameter fine-tuning:

$$\min_{\phi,\, \theta_r} \; - \sum_{(x,y) \in \mathcal{D}} \sum_{t=1}^{N} \log p_{\theta_r, \phi}\big(y_t \mid x, y_{<t}, \mathcal{M}_t\big), \tag{12}$$

where $\mathcal{M}_t$ follows the mask of Equation 10 and review boundaries are sampled per example from the inference schedule. Under teacher forcing, prefix determinism makes every review message a slice of the bridge output on $x \oplus y$, so a single sender prefill yields all messages and reviews enter training only through the mask. Zero-initialized gates make the adapted receiver identical to the original at the start of training. Full-parameter tuning of the receiver consistently outperforms training the bridge alone.

4 EXPERIMENTS

We evaluate POND in two settings: a single prefill-only transfer, where the sender reads the input once, and periodic review, where the sender additionally reads the receiver's partial response at selected boundaries.

4.1 EXPERIMENTAL SETUP

Models and comparison conditions. Qwen3.5-4B (Team, 2026) is the receiver in all conditions. The senders are Qwen3.8-Max (Qwen Team, 2026), GLM-5.3 (GLM-5-Team et al., 2026), and DeepSeek-V4-Flash (DeepSeek-AI, 2026), providing one within-family and two cross-family pairings. The sender remains frozen, while the receiver and bridge are jointly optimized using the conditional likelihood in Equation 1. *Receiver alone* generates from the original prompt without the bridge. POND supplies latent KV entries from one sender prefill pass and makes no further sender calls. *+Review* additionally processes the available response prefix and appends latent updates as described in Section 3.5. In both bridged conditions, all output tokens are generated by the receiver. The receiver-alone comparison measures the combined effect of receiver adaptation, bridge training, and sender guidance, so it does not isolate the contribution of sender information from that of receiver training. The POND–Review comparison measures the additional benefit and cost of periodic sender updates under matched training and evaluation protocols.

Tasks and quality metrics. We report AIME (2022-2026) (Mathematical Association of America) and OlymMath-Hard (Sun et al., 2026a) for mathematics; HumanEval+, MBPP+ (Liu et al., 2023; EvalPlus Team, 2023), and LiveCodeBench v6 (Jain et al., 2025) for code; and GPQA-Diamond (Rein et al., 2024), MMLU-Redux (Gema et al., 2025), MMLU-Pro (Wang et al., 2024), and ARC-Challenge (Clark et al., 2018) for knowledge-intensive question answering. Acc. denotes exact-match accuracy for math and multiple-choice tasks and pass@1 for code, in percent. Avg is the unweighted mean of task scores within each suite, not accuracy pooled over examples.

Efficiency metrics and aggregation. Token reports decoded output in thousands of tokens per 100 runs; an entry of v corresponds to $10v$ tokens per run. It excludes sender prefill tokens and latent KV memory and is not a measure of total FLOPs. Latency reports end-to-end time in seconds per run, including sender prefill, bridge computation, and receiver decoding; lower is better. Suite-level token and latency reductions are calculated from the unweighted per-task means as $100(1 - m/m_{\text{receiver}})$. This ratio of means differs from the mean of task-wise percentage reductions and gives greater influence to tasks with larger baseline costs. Accuracy gains are absolute percentage-point differences, calculated before rounding. The Improve columns select the better bridged value separately for each metric; the joint comparisons below instead use explicitly named configurations.

4.2 MAIN RESULTS ON MATH AND CODE

Single-transfer guidance accounts for most of the accuracy gain. Table 1 reports higher accuracy than the receiver for all 15 sender–task pairs even without review. From the receiver's 63.94% macro-average, single-transfer POND reaches 67.14%, 66.74%, and 66.2% with Qwen3.8-Max, GLM-5.3, and DeepSeek-V4-Flash, respectively. Review raises these to 68.02%, 67.42%, and 66.9%, improvements of 4.08, 3.48, and 2.96 points over the receiver. Single-transfer POND accounts for roughly 80% of the final gain, so most of the improvement does not require repeated sender calls.

Task-level gains are heterogeneous. Table 1 shows the largest +Review gains on AIME (5.1–6.2 points) and LiveCodeBench (4.2–5.0), compared with 1.9–2.6 on HumanEval+ and 2.0–2.9 on

Table 1: **Math and code results.** Senders are frozen and prefill-only. POND uses one latent transfer; +Review adds periodic prefill updates. Acc.: accuracy/pass@1 (%, higher is better); Token: output volume (k/100 runs, lower is better); Speed: end-to-end latency (s/run, lower is better). Bold marks the better bridged value per metric, including ties. Improve reports accuracy gains in percentage points or relative token/latency reductions against the receiver; upward latency arrows indicate increases. Metric-wise selections need not describe one configuration. Avg averages the five tasks.

Tasks	Metrics	Receiver alone	Sender: Qwen3.8-Max			Sender: GLM-5.3			Sender: DeepSeek-V4-Flash		
			POND	+Review	Improve	POND	+Review	Improve	POND	+Review	Improve
AIME	Acc.	80.0	85.4	**86.2**	↑ 6.2	84.9	**85.5**	↑ 5.5	84.6	**85.1**	↑ 5.1
	Token	982.4	836.7	**768.9**	↓ 21.7%	861.5	**801.4**	↓ 18.4%	879.2	**829.7**	↓ 15.5%
	Speed	219.4	**190.7**	199.6	↓ 13.1%	**196.8**	205.1	↓ 10.3%	**202.3**	209.7	↓ 7.8%
OlymMath Hard	Acc.	50.0	52.9	**53.7**	↑ 3.7	52.6	**53.1**	↑ 3.1	52.2	**52.8**	↑ 2.8
	Token	1437.8	1211.6	**1094.3**	↓ 23.9%	1248.9	**1152.7**	↓ 19.8%	1281.4	**1197.6**	↓ 16.7%
	Speed	311.7	**263.8**	276.4	↓ 15.4%	**271.6**	282.3	↓ 12.9%	**280.9**	290.1	↓ 9.9%
HumanEval+	Acc.	81.7	83.5	**84.3**	↑ 2.6	83.2	**83.9**	↑ 2.2	83.0	**83.6**	↑ 1.9
	Token	94.6	86.1	**81.9**	↓ 13.4%	88.0	**84.7**	↓ 10.5%	89.5	**86.8**	↓ 8.2%
	Speed	25.9	**24.1**	26.0	↓ 6.9%	**24.7**	26.6	↓ 4.6%	**25.2**	27.0	↓ 2.7%
MBPP+	Acc.	64.8	66.8	**67.7**	↑ 2.9	66.4	**67.42**	↑ 2.4	66.1	**66.8**	↑ 2.0
	Token	57.9	52.8	**50.1**	↓ 13.5%	54.1	**52.2**	↓ 9.8%	55.0	**53.6**	↓ 7.4%
	Speed	16.8	**16.0**	17.3	↓ 4.8%	**16.4**	17.8	↓ 2.4%	**16.6**	18.1	↓ 1.2%
LiveCode Bench	Acc.	43.2	47.1	**48.2**	↑ 5.0	46.6	**47.4**	↑ 4.2	45.2	**46.1**	↑ 4.6
	Token	371.5	319.7	**289.8**	↓ 22.0%	327.6	**305.4**	↓ 17.8%	338.2	**316.7**	↓ 14.8%
	Speed	88.3	**75.6**	79.1	↓ 14.4%	**78.9**	82.0	↓ 10.6%	**81.2**	84.6	↓ 8.0%
Avg	Acc.	63.94	67.14	**68.02**	↑ 4.08	66.74	**67.42**	↑ 3.48	66.2	**66.9**	↑ 2.96
	Token	588.8	501.4	**457.0**	↓ 22.4%	516.0	**479.3**	↓ 18.6%	528.7	**496.9**	↓ 15.6%
	Speed	132.4	**114.0**	119.7	↓ 13.9%	**117.7**	122.8	↓ 11.1%	**121.2**	125.9	↓ 8.4%

MBPP+. OlymMath-Hard improves by 2.8–3.7 points despite a baseline of only 50.0%, less than the gain on AIME with its 80.0% baseline. Baseline headroom alone therefore does not explain the variation. Overall, review consistently improves accuracy across these benchmarks, although the magnitude of the gains varies by task.

The pattern extends across sender families. Both cross-family senders improve the receiver on every math/code task. Qwen3.8-Max with review has the highest macro-average and leads on all five tasks. The cross-family results show that the gains are not confined to a within-family pairing, but sender-alone scores and uncertainty estimates are needed to compare transfer efficiency reliably.

4.3 Results on Knowledge Benchmarks

Gains extend beyond the math/code suite. Table 2 reports improvements for all 12 sender–task pairs. From a receiver average of 84.77%, single-transfer POND reaches 86.44%, 86.08%, and 85.75% with Qwen3.8-Max, GLM-5.3, and DeepSeek-V4-Flash. Review increases these to 87.18%, 86.68%, and 86.35%, or 1.58–2.41 points above the receiver. The ordering of senders by macro-average matches that in math/code, but the absolute improvements are smaller.

Review sensitivity varies by task. Review adds 1.01–1.52 points on GPQA-Diamond, the longest-output knowledge task, compared with 0.35–0.62 points on the other three tasks. This pattern motivates evaluating review's benefit together with its latency cost rather than assuming that repeated sender involvement is equally useful across tasks.

4.4 Accuracy–Latency Trade-offs and Periodic Review

Shorter outputs do not imply proportional latency savings. Relative to the receiver, +Review reduces average output volume by 15.6–22.4% on math/code and 15.5–22.3% on knowledge tasks (Tables 1 and 2). The corresponding latency reductions are only 4.9–9.6% on math/code and range from a 6.5% reduction to a 0.1% increase on knowledge tasks. Sender prefill and latent conditioning still incur computation, so fewer receiver tokens do not establish proportional savings in total compute.

Table 2: **Knowledge benchmark results.** Configurations, metrics, and metric-wise bolding follow Table 1. Improve reports accuracy gains (percentage points) or relative token/latency reductions (%) against the receiver; upward latency arrows denote increases. Avg averages all four tasks. Differences and reductions are computed before rounding.

Tasks	Metrics	Receiver alone	Sender: Qwen3.8-Max			Sender: GLM-5.3			Sender: DeepSeek-V4-Flash		
			POND	+Review	Improve	POND	+Review	Improve	POND	+Review	Improve
GPQA Diamond	Acc.	76.77	79.29	**80.81**	↑ 4.04	78.79	**79.80**	↑ 3.03	78.28	**79.29**	↑ 2.52
	Token	660.3	568.9	**504.7**	↓ 23.6%	585.1	**529.8**	↓ 19.8%	601.4	**548.6**	↓ 16.9%
	Speed	149.2	**128.6**	136.9	↓ 13.8%	**134.8**	142.1	↓ 9.7%	**139.7**	146.5	↓ 6.4%
MMLU Redux	Acc.	89.58	90.47	**90.86**	↑ 1.28	90.22	**90.63**	↑ 1.05	90.04	**90.39**	↑ 0.81
	Token	102.4	95.6	**89.7**	↓ 12.4%	97.1	**92.4**	↓ 9.8%	98.5	**94.9**	↓ 7.3%
	Speed	27.1	**25.7**	28.1	↓ 5.2%	**26.3**	28.8	↓ 3.0%	**26.8**	29.4	↓ 1.1%
MMLU Pro	Acc.	78.62	80.93	**81.47**	↑ 2.85	80.51	**81.08**	↑ 2.46	80.12	**80.74**	↑ 2.12
	Token	286.7	246.2	**218.5**	↓ 23.8%	253.8	**229.7**	↓ 19.9%	262.1	**239.6**	↓ 16.4%
	Speed	69.4	**60.5**	64.3	↓ 12.8%	**63.4**	67.1	↓ 8.6%	**65.7**	69.2	↓ 5.3%
ARC Challenge	Acc.	94.11	95.05	**95.56**	↑ 1.45	94.80	**95.22**	↑ 1.11	94.54	**94.97**	↑ 0.86
	Token	145.8	127.9	**116.3**	↓ 20.2%	131.6	**121.5**	↓ 16.7%	135.2	**126.8**	↓ 13.0%
	Speed	37.6	**33.1**	35.6	↓ 12.0%	**34.5**	37.0	↓ 8.2%	**35.7**	38.4	↓ 5.1%
Avg	Acc.	84.77	86.44	**87.18**	↑ 2.41	86.08	**86.68**	↑ 1.91	85.75	**86.35**	↑ 1.58
	Token	298.8	259.7	**232.3**	↓ 22.3%	266.9	**243.4**	↓ 18.6%	274.3	**252.5**	↓ 15.5%
	Speed	70.8	**62.0**	66.2	↓ 12.5%	**64.8**	68.8	↓ 8.6%	**67.0**	70.9	↓ 5.4%

Review improves quality more consistently than latency. Review improves accuracy and reduces output volume relative to single-transfer POND in all 27 sender–task pairs. Its incremental macro-average gain is 0.7–0.9 points on math/code and 0.60–0.74 points on knowledge tasks. However, the additional prefill passes increase latency relative to single transfer in every pair. Review remains faster than the receiver on the longest-output tasks, including AIME, OlymMath-Hard, LiveCodeBench, and GPQA-Diamond, but exceeds receiver latency for every sender on the shorter HumanEval+, MBPP+, and MMLU-Redux tasks. This pattern is consistent with review overhead being easier to amortize over longer generations, although component-level timing is needed to establish the mechanism.

The benefit of review is consistent with a trajectory-dependent effect. Its incremental gains are largest on long-output tasks such as AIME, LiveCodeBench, and GPQA-Diamond, where guidance computed from the initial prompt may become less informative as the response develops. On shorter tasks, review provides smaller accuracy gains while adding proportionally greater latency. This pattern suggests that review is most useful when the evolving reasoning trajectory contains information not available in the initial prompt.

Aggregate improvements are not uniform per-task speedups. All three review configurations improve mean accuracy and latency over the receiver on math/code. On knowledge tasks, Qwen3.8-Max and GLM-5.3 improve both means, while DeepSeek-V4-Flash improves accuracy but is 0.1 seconds slower on average. Because longer tasks contribute more to the ratio-of-means latency summary, suite-level improvements do not imply a speedup for every workload or justify a fixed review policy under every latency constraint. Qwen3.8-Max has the best aggregate accuracy, output volume, and latency in both suites, although these results do not establish a general ranking of sender quality.

Table 3: **Training and review controls.** Suite-average accuracy. Receiver-only training uses math/code data; POND uses Qwen3.8-Max as the sender.

Method	Math/code	Knowledge
Frozen receiver	63.94	84.77
Receiver-only training	64.2	83.45
POND	66.9	86.44
POND + Review	**68.02**	**87.18**

Receiver-only training does not explain the full gains. As shown in Table 3, training the receiver directly on math/code raises its suite average from 63.94% to 64.2%, but lowers knowledge accuracy from 84.77% to 83.45%, suggesting a small loss in general-knowledge performance. Jointly training the receiver and bridge with latent guidance improves both suites, reaching 66.9% on math/code and 86.44% on knowledge with a single transfer, with further gains from review.

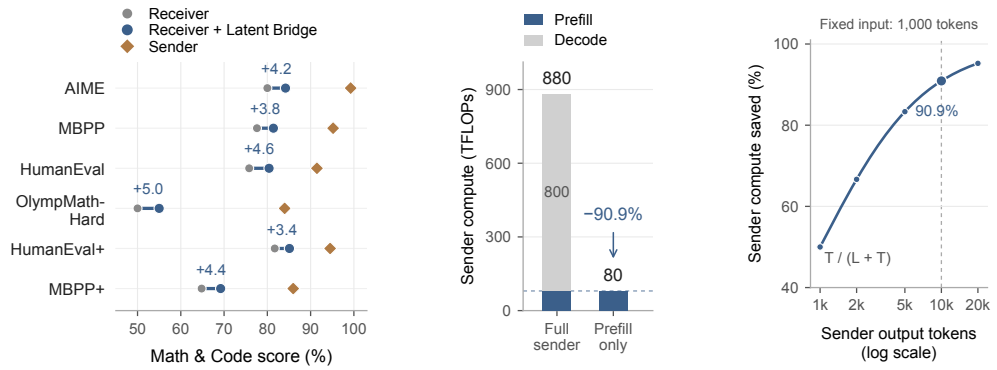

Figure 2: **Accuracy gains and sender compute savings. Left:** Math and code scores for the receiver (Qwen3.5-4B), the receiver with POND, and the sender (GLM-5.3); labels give the gain in points. **Center:** Estimated sender FLOPs for GLM-5.3 (40B active) with 1k input and 10k output tokens. Prefill-only guidance reduces sender compute from 880 to 80 TFLOPs. **Right:** Fraction of sender compute saved, $T/(L+T)$, as a function of output length T with $L = 1$k. Estimates use $2P_{\text{active}}$ FLOPs per token and exclude attention, routing, communication, and bridge or receiver inference. The panels are not a matched end-to-end comparison of accuracy and cost.

4.5 Accuracy Gains and Sender Compute

Figure 2 relates the accuracy gained from latent guidance to the sender-side computation avoided relative to textual advising. The panels provide complementary views of quality and analytical compute rather than matched end-to-end operating points. We highlight three observations.

A single transfer recovers part of the sender advantage. Across the benchmarks in the left panel, POND consistently improves the receiver, but closes different fractions of the receiver–sender gap. The recovered fraction is smaller on tasks where the sender's advantage is larger. This suggests that prompt representations convey useful task information without reproducing all of the computation performed during sender-side generation.

Review helps most on longer trajectories. Periodic review yields its largest gains on long-output tasks (AIME, LiveCodeBench, GPQA-Diamond) and smaller gains on shorter ones. One explanation is that guidance derived from the initial prompt grows stale as the response unfolds, whereas review conditions on the intermediate trajectory. Aggregate results, however, cannot separate better planning from error correction or reduced verbosity.

Sender-side savings scale with the avoided response length. Under a parameter-multiply approximation, sender generation costs $2P_{\text{active}}(L+T)$ versus $2P_{\text{active}}L$ for a single prefill, saving a fraction $T/(L+T)$. With cached delta-only review, this becomes $(T-\sum_r \Delta U_r)/(L+T)$, positive whenever $\sum_r \Delta U_r < T$. These estimates exclude attention, communication, and receiver inference, as well as review overheads, and thus reflect sender-side savings rather than end-to-end speedup.

5 Conclusion

We introduced POND, which decouples frontier-model prefill from small-model decoding through a learned latent interface. A trainable bridge converts a frozen sender's final-layer representations into latent KV entries, from which a receiver generates the full answer without any textual handoff. Periodic review extends this to partial responses, refreshing guidance while preserving the receiver's generation state. The central contribution is thus an interface through which a large model participates in generation without decoding text itself. Establishing its advantage over other collaboration strategies will require matched-compute text baselines, controlled probes of sender information, and uncertainty estimates; adaptive review policies and bounded latent memory are natural next steps.

AI USE STATEMENT

We used generative AI tools to assist with drafting and editing the manuscript, general code editing, and preparing figures and charts. We take responsibility for the final content of this work, including all text, code, and figures produced with AI assistance.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024.

Justin Chen, Swarnadeep Saha, and Mohit Bansal. Reconcile: Round-table conference improves reasoning via consensus among diverse llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7066–7085, 2024.

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, 2018. URL https://arxiv.org/abs/1803.05457.

DeepSeek-AI. DeepSeek-V3.2: Pushing the frontier of open large language models, 2025. URL https://arxiv.org/abs/2512.02556.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression, 2026. URL https://arxiv.org/abs/2609.19969.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

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.

EvalPlus Team. EvalPlus, 2023. URL https://github.com/evalplus/evalplus. MBPP+ dataset version 0.2.0 (378 tasks); accessed 2026-07-25.

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 *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=LeatkxrBCi.

Aryo Pradipta Gema, Joshua Ong Jun Leang, Giwon Hong, Alessio Devoto, Alberto Carlo Maria Mancino, Rohit Saxena, Xuanli He, Yu Zhao, Xiaotang Du, Mohammad Reza Ghasemi Madani, Claire Barale, Robert McHardy, Joshua Harris, Jean Kaddour, Emile Van Krieken, and Pasquale Minervini. Are we done with MMLU? In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5069–5096, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.262. URL https://aclanthology.org/2025.naacl-long.262/.

GLM-5-Team, :, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can

Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiaxin Zhang, Jiayi Gui, Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhan Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

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

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. In *International Conference on Learning Representations*, volume 2024, pp. 27896–27923, 2024.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. In *International Conference on Learning Representations*, volume 2024, pp. 32694–32717, 2024.

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

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the association for computational linguistics: ACL 2023*, pp. 8003–8017, 2023.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. In *Findings of the association for computational linguistics: EMNLP 2020*, pp. 4163–4174, 2020.

Yoon Kim and Alexander M Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 1317–1327, 2016.

Yunxuan Li, Yibing Du, Jiageng Zhang, Le Hou, Peter Grabowski, Yeqing Li, and Eugene Ie. Improving multi-agent debate with sparse communication topology. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 7281–7294, 2024.

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

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and LINGMING ZHANG. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 21558–21572. Curran Associates, Inc., 2023. doi: 10.52202/075280-0943. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/43e9d647ccd3e4b7b5baab53f0368686-Paper-Conference.pdf.

Mathematical Association of America. American Invitational Mathematics Examination (AIME). https://maa.org/maa-invitational-competitions/. Accessed: 2026-09-26.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative LLM inference using phase splitting. In *Proceedings of the 51st Annual International Symposium on Computer Architecture (ISCA)*, June 2024. URL https://www.microsoft.com/en-us/research/publication/splitwise-efficient-generative-llm-inference-using-phase-splitting/.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. 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.

Tingyu Qu, Tinne Tuytelaars, and Marie-Francine Moens. Introducing routing functions to vision-language parameter-efficient fine-tuning with low-rank bottlenecks. In *European Conference on Computer Vision*, pp. 291–308. Springer, 2024.

Qwen Team. Qwen3.8-max: A new bar for coding and cowork, August 2026. URL https://qwen.ai/blog?id=qwen3.8.

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. URL https://openreview.net/forum?id=Ti67584b98.

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 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 677–693, 2025.

Haoxiang Sun, Yingqian Min, Zhipeng Chen, Xin Zhao, and Ji-Rong Wen. Challenging the boundaries of reasoning: An olympiad-level math benchmark for large language models. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17438–17457, San Diego, California, United States, July 2026a. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.792. URL https://aclanthology.org/2026.acl-long.792/.

Wei Sun, Tingyu Qu, Jesse Davis, and Marie-Francine Moens. Event interaction in low-rank bottlenecks for temporal relation extraction. *arXiv preprint arXiv:2609.06731*, 2026b. URL https://arxiv.org/abs/2609.06731.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities, 2024. *URL https://arxiv.org/abs/2406.04692*, 1(2):5.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems*, 33:5776–5788, 2020.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-pro: A more robust and challenging multi-task language understanding benchmark. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=y10DM6R2r3.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023. *URL https://arxiv. org/abs/2308.08155*, 2308, 2023.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin.

A Appendix

This appendix documents the implementation and training semantics used in the experiments reported in Section 4. The completed system used final-layer sender features, a frozen sender, joint receiver/bridge training, dense latent attention, and cached delta-only review. All numerical hyperparameters were tied to the corresponding run manifests, and all benchmark measurements reported in the paper were produced by these runs.

A.1 Objective and gradient paths

Equation 12 averaged response-token negative log-likelihood over valid targets. Prompt and padding positions received zero loss. Sender features were detached, while receiver operations remained differentiable with respect to both receiver weights and injected KV entries. The trained bridge consisted of the two-layer compressor, per-layer K/V projections, key RMS-normalization scales, and scalar gates. The dense configuration used in the main experiments had no indexer parameters.

For the bridge-only ablation, we kept the receiver weights fixed while retaining autograd through receiver operations to the bridge. For the no-message jointly trained receiver ablation, we optimized the receiver without latent messages. We assigned distinct condition names and artifact hashes to the original receiver, the receiver-only training baseline, the bridge-only ablation, and the jointly trained receiver/bridge system.

Teacher-forced review. For each fixed tokenized input/answer sequence, the causal sender computed all features in one full pass. Review r selected only the features associated with its newly observed segment. The training attention mask enforced both span alignment and review release, preventing review features from becoming visible to earlier receiver queries. Consequently, the receiver never had access to future-answer information during training. Single-pass feature extraction reduced training cost without changing the causal message mask.

Train–inference equivalence. We validated full causal sender execution against cached chunk execution on identical token sequences and boundary schedules. The comparison used matched positions, tokenization, architecture, and numerical settings. We likewise validated receiver full-pass execution with per-query masks against incremental execution with preserved historical states. The two paths agreed within the numerical tolerance defined in the run manifest. This validation does not imply bitwise equality under arbitrary kernels: dropout, implementation-dependent floating-point order, and stateful kernels can alter the result. Moreover, teacher forcing uses reference prefixes whereas deployment uses sampled prefixes, so the validation does not remove the resulting distribution shift.

B Position Alignment and Message Availability

We treated the position identifier and release threshold as separate quantities. The former controlled rotary phase, whereas the latter prevented a receiver query from reading an unavailable message. For a shared serialized text, Equation 9 assigned each sender token to the first receiver token covering its complete source span. Multiple sender tokens could map to one receiver position; they remained distinct KV entries and did not consume or shift text-cache positions.

Offset convention. We represented offsets in a common UTF-8 byte coordinate system and converted tokenizer-provided offsets before alignment. Sender and receiver offsets were computed from the same normalized and serialized string, including recorded chat delimiters and special tokens. We excluded non-text tokens without an unambiguous receiver counterpart from latent memory rather than assigning guessed spans. Inputs whose normalization or serialization could not be reconciled were marked as interface failures and excluded according to the predefined evaluation rule.

Incremental tokenization. At every review, the implementation verified that the new sender token sequence extended the cached token prefix. We used a prefix-stable streaming tokenization policy and retained token IDs and source spans at every boundary. When a cache fallback was required and the existing tokens and positions were unchanged, the implementation recomputed the sender

prefix, charged the full recomputation, and appended only newly available states. If old tokens or source spans changed, the implementation recorded a tokenization failure and terminated the request rather than silently altering receiver history. Full request restarts, where applicable, were recorded separately together with their work and generated output.

Causal visibility claim. Assume (i) sender state j depends only on its fixed serialized source prefix; (ii) alignment p_j covers that complete prefix span; (iii) release decisions use only already observed text and runtime state; and (iv) the receiver is causal and never recomputes historical states with later messages. Then prediction of y_t under Equation 10 cannot depend on y_t or later response tokens. Indeed, its query position is $q = n_x + t - 1$; every visible latent satisfies $p_j \le q$ and $\rho_r \le q$, so its source and release are both in the observed prefix. Text attention and recurrent states are also causal. Induction through receiver layers and decoding steps proves the claim. This is a conditional correctness property, not an accuracy theorem.

Next-token indexing. The receiver query at $q = n_x + t - 1$ predicted y_t. A review issued after sampling y_b had release position $\rho = n_x + b$ and first affected prediction of y_{b+1}. For each latent entry j, the implementation masked the entry unless $q \ge \max(p_j, \rho)$. This rule also applied when its rotary position was early in the prompt. Padding was never visible. Receiver prompt queries used the same positional causal mask and therefore did not attend to the full prompt message prematurely.

C Gated Attention and Sparse Reading

For one query/head, let $C_{\mathrm{txt}} = \sum_i \exp(a_i^{\mathrm{txt}})$ and $C_z = \sum_j \exp(a_j^z)$ sum valid scaled logits. With separately normalized outputs A^{txt} and A^z,

$$A^{\mathrm{joint}} = (1-\alpha)A^{\mathrm{txt}} + \alpha A^z, \qquad \alpha = \frac{C_z}{C_{\mathrm{txt}} + C_z}. \tag{13}$$

This proves Equation 7 before dropout. Our implementation combined log-sum-exp statistics rather than exponentiating large raw logits. When latent memory was empty, it returned text attention. Duplicate messages were not treated as neutral because adding entries changes the normalizer.

Zero-gate initialization. At $g_\ell = 0$, the local attention equals the original receiver output. The gate derivative is

$$\frac{\partial A_\ell}{\partial g_\ell} = (1 - \tanh^2 g_\ell)(A_\ell^{\mathrm{joint}} - A_\ell^{\mathrm{txt}}). \tag{14}$$

The latent K/V gradient was initially multiplied by zero, while the gate could receive a nonzero gradient. With all gates initialized to zero, the compressor received no likelihood gradient on the first step in the absence of another loss path. Receiver weights continued to receive ordinary language-model gradients under joint training. Subsequent optimization was allowed to move the receiver away from the original checkpoint.

Sparse variant. The dense configuration retained every visible entry and used no sequence compression ($m = 1$). In the completed sparse evaluation, top-k reading applied Equation 11 per query head after masking, with stable entry-index tie breaking. When fewer than k entries were visible, the implementation retained all of them; when none were visible, it returned text attention. The latent weighted sum and its log-normalizer used only retained entries, while text entries were never pruned. Exact query–key scoring still touched every visible latent key, so top-k reduced selected-value aggregation rather than making total runtime independent of memory size. We trained the reference model densely and evaluated sparse deployment in a separate quality/cost sweep.

Truncation error at a fixed query. Suppose the text and latent values have norm at most B. Let δ be the probability mass that the *full joint softmax* assigns to latent entries omitted by sparse reading. Denote the dense and sparse gated outputs by A and $\widetilde{A}$, using the same gate and text branch. Conditioning the full softmax on retained entries gives $A^{\mathrm{joint}} = (1-\delta)\widetilde{A}^{\mathrm{joint}} + \delta A^{\mathrm{omit}}$. Therefore

$$\|A - \widetilde{A}\| \le 2B|\tanh g|\,\delta. \tag{15}$$

Algorithm 1 Inference with POND

Require: input x; review boundaries $b_1 < \cdots < b_R$; output limit $N_{\max}$

Set $b_0 \leftarrow 0$, $r \leftarrow 1$, and $n_x \leftarrow |x|_{\text{receiver}}$
Serialize x; record sender token IDs, spans, and aligned positions
Sender prefill on x (cache kept); build $\mathcal{M}^{(0)}$ from $Z^{(0)} = B_\phi(x)$
Set $\mathcal{M} \leftarrow \mathcal{M}^{(0)}$
Receiver prefill on x with $\mathcal{M}^{(0)}$ and Equation 10
for $t = 1, 2, \ldots, N_{\max}$ **do**
    Sample y_t from the receiver
    if $y_t = \texttt{EOS}$ or $t = N_{\max}$ **then**
        break
    end if
    if $r \leq R$ and $t = b_r$ **then**
        Serialize available text; verify unchanged sender prefix IDs and spans
        if prefix stability or valid position alignment fails **then**
            Record interface failure and **terminate** the request
        end if
        Sender prefill on $y_{b_{r-1}+1:b_r}$ using its retained cache
        Build $Z^{(r)}$ from new states; append its KV entries to $\mathcal{M}$ (Equation 8); set $\rho_r = n_x + t$
        $r \leftarrow r + 1$
    end if
    Receiver processes y_t at position $n_x + t$ with $\mathcal{M}$ (Equation 7; optionally Equation 11)
end for

Both conditional weighted sums have norm at most B, which establishes the inequality by the triangle inequality. If nothing is omitted, the outputs agree. A small k alone does not bound δ, so sparse reading has no unconditional approximation guarantee. This bound is local to one attention output; later layers and autoregressive sampling may amplify a perturbation. It does not bound answer accuracy, and estimating the exact omitted mass itself requires full scoring.

D Inference and Cache Semantics

Algorithm 1 specifies the incremental-prefill inference procedure used in our evaluation. Here $B_\phi(x)$ abbreviates compression of sender features, and $\mathcal{M}$ denotes the collection of per-layer memories. Sender input segments were serialized using the procedure in Appendix B.

Historical states were not rewritten. Each cached receiver state recorded the messages available when it was computed. Reviews updated future queries without changing historical states. Thus, the implementation was causal by induction on decoding steps but was not equivalent to recomputing the entire prefix with the newest memory visible at all positions. Hybrid receiver state, including non-attention recurrent state, was preserved together with the text KV cache. Reviews did not retract tokens that had already been emitted.

Delta-only versus full-message review. The evaluated implementation appended each new sender token once. Re-prefilling and appending the entire context at every boundary would duplicate earlier latent entries and alter attention normalization and storage; we did not use that variant in Algorithm 1. When a cache fallback recomputed old sender states, it appended only new entries and recorded full-prefix sender work separately from delta-only receiver storage.

E Resource Accounting

E.1 Sender work

Let P_s be sender active parameters, L_s its prompt-token count, D_s the generated-token count of a hypothetical sender-decoding baseline, and ΔU_r the newly processed sender tokens in review r. Under a parameter-multiply-only approximation,

$$F_{\text{sender,decode}} \approx 2P_s(L_s + D_s), \tag{16}$$

$$F_{\text{sender,initial}} \approx 2P_s L_s, \tag{17}$$

$$F_{\text{sender,cached}} \approx 2P_s \left(L_s + \sum_r \Delta U_r \right). \tag{18}$$

The corresponding avoided fraction is

$$\eta_{\text{cached}} \approx \frac{D_s - \sum_r \Delta U_r}{L_s + D_s}. \tag{19}$$

Savings are positive only when reviewed sender tokens are fewer than the avoided sender output tokens. This comparison is against sender decoding, not receiver-alone inference. For $P_s = 40$ billion, $L_s = 1000$, $D_s = 10000$, and no reviews, the analytical approximation gives 80 versus 880 TFLOPs, or 90.9% avoided parameter-multiply work. We treated these as analytical estimates rather than GPU measurements.

For recorded no-cache fallbacks, a full sender context of length U_r was reprocessed at each review, costing $2P_s(L_s + \sum_r U_r)$. Under common-token boundaries $b_r = rq$, $\sum_r U_r \approx RL_s + qR(R + 1)/2$. We applied this quadratic full-prefill expression only to no-cache fallbacks. Cached review still attended to the retained sender prefix and preserved sender state; we therefore did not equate token-pass counts with wall-clock time across chunk sizes, attention kernels, or devices.

E.2 Latent memory, communication, and attention

Let V_0 be the number of prompt sender tokens retained as valid latent entries and ΔV_r the retained entries in review r. Unlike the processed lengths L_s and ΔU_r, these excluded unmapped special tokens. Delta-only storage was $M_z = V_0 + \sum_r \Delta V_r \le L_s + \sum_r \Delta U_r$; equality held only when every processed token was retained. Sender FLOPs included all processed tokens, including excluded special tokens. With L_a adapted layers and β_{KV} bytes per scalar, added receiver KV storage was

$$B_{\text{latent KV}} = 2\beta_{\text{KV}} L_a M_z n_{\text{kv}} d_h. \tag{20}$$

This quantity excluded sender caches, text caches, weights, and temporary activations. For nonuniform KV layouts, we summed the corresponding quantity over layers. Sharing Z reduced one-way transmission to $\beta_z M_z d_z$ bytes when compression ran sender-side and KV projection ran receiver-side; it did not remove the receiver's per-layer cache. Review also transmitted text-prefix increments in the reverse direction.

For n_q receiver query heads and $M_z(t)$ visible latent entries, dense response attention added approximately

$$F_{\text{latent attention}} \approx 4L_a n_q d_h \sum_{t=1}^{N} M_z(t), \tag{21}$$

excluding softmax, projections, prompt attention, and duplicated text work. KV storage scaled with n_{kv}, not n_q. With exact top-k selection, scoring remained linear in $M_z(t)$ while only selected-value aggregation scaled with k. More frequent reviews could increase the sum even at a fixed final memory length because entries became visible earlier.

End-to-end accounting. For the serial implementation, we decomposed latency as

$$\tau = \tau_s + \tau_{\text{bridge}} + \tau_{\text{transfer}} + \tau_{r,\text{prefill}} + \tau_{r,\text{decode}}. \tag{22}$$

The sender term included every review and cache fallback. Where stages overlapped, we measured the critical path instead of adding overlapping intervals. We assessed speedup against receiver-alone inference only when the measured reduction in receiver work exceeded all added overheads. We did not treat shorter output length alone as a total-FLOPs or latency measurement.

F Reproducibility and Scope

Configuration status. The completed run manifests recorded model and tokenizer revisions, hidden-state access, bridge dimensions, adapted layers, normalization, gate initialization, training data and splits, optimizer and schedule, precision, random seeds, checkpoint selection, review policy, and decoding settings. All aggregate tables were generated from per-run artifacts associated with these manifests rather than reconstructed from summary scores. Sender models were accessed through interfaces exposing the required hidden states; no text-only sender API was used for the latent interface.

F.1 Experimental Configuration

The machine-readable configuration file `configs/reference_design.json` records the configuration used in the completed runs. We used $d_b = 2048$, $d_z = 512$, key RMSNorm epsilon 10^{-6}, dense attention without dropout, and adapters in all full-attention receiver layers. We optimized with AdamW using receiver and bridge learning rates of 5×10^{-6} and 10^{-4}, betas $(0.9, 0.95)$, weight decay 0.01, gradient clipping at 1, an effective batch size of 32 examples, one epoch, and 3% warm-up followed by cosine decay. Norms and loss used FP32, and model computation used BF16.

We trained with reference-prefix reviews every 512 receiver tokens. At the same checkpoint, we evaluated no-review inference and review strides of 256, 512, and 1024 tokens. Evaluation used greedy generation with one output per problem and a 4096-token output cap within an 8192-token combined text limit. Oversize examples were identified and rejected using the predefined rule rather than silently truncated. We completed independent training runs with seeds 11, 23, and 47. On the densely trained model, we also completed the sparse sweep with $k \in \{32, 64, 128\}$ and $m = 1$.

The run manifests record the exact model and tokenizer revisions, training and evaluation dataset manifests, licenses, software environment, and hardware configuration. Pre-run validation confirmed hidden-state access and agreement between the implementation and the equations; completion records and checkpoint hashes were stored for every successful training run.

Evaluation record. For every benchmark, we recorded the version, sample count, prompt template, reasoning mode, answer extraction procedure, executable-test settings and timeouts, truncation and failure handling, and decontamination procedure. We archived per-example predictions, correctness, latency, output length, review count, sender-token count, cache-fallback count, and peak memory. Timing records included hardware, software, parallelism, concurrency, warm-up, synchronization, repetition count, and whether loading, queueing, and communication were included.

Attribution and uncertainty. The completed comparison included the original receiver, a data- and budget-matched receiver-only training baseline, and the jointly trained receiver/bridge system. We additionally evaluated the frozen-receiver bridge ablation, constant and shuffled messages, and matched-budget text advising as separate controls. Review-frequency comparisons used the same receiver/bridge checkpoint. Paired uncertainty estimates resampled the same problems across systems and grouped multiple samples from each problem. Training-seed variation used the three completed independent runs, and timing variation was estimated from repeated synchronized measurements.

Limitations. Hidden-state access limits black-box deployment. Incremental review requires prefix-stable tokenization, available sender state, and custom receiver kernels. Compressed states are not automatically private or safe to transmit. Review affects only future predictions and cannot certify or repair past outputs. The reported downstream accuracy and efficiency remain specific to the evaluated models, datasets, configurations, and comparison baselines; they do not establish a universal advantage over all collaboration methods.