RECEIVER-CONDITIONED LATENT COMMUNICATION GIVES 94% CACHEBACK

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent systems distribute large contexts across agents that communicate to solve a task. Text messages are compact but require decoding and may omit evidence the receiving agent needs. Recent latent communication instead transfers KV caches. This avoids text generation and can improve accuracy and latency. However, a full KV cache grows linearly with both the context an individual agent processes, and the number of agents that coordinate together. This raises memory and context costs, often far exceeding available GPU resources and context window sizes. Our key observation is that agents need only send what the receiving agent requires for its local task—which we call *receiver-conditioned communication*. The receiver agent passes the sender a small description of its information needs, which serves to filter and compress the sender agent's KV cache. **CacheBack** is a simple, robust, training-free instance of *receiver conditioning* based on the sender's attention weights. On FanOutQA, CacheBack with Qwen 3 removes 75% of the state the agent would otherwise receive, improving accuracy by 14.7 percentage points and reducing median task-completion latency by 3.2× relative to text communication. We show comparable improvements across model families that span dense Transformers, Mamba-attention hybrids, and sliding-window attention.

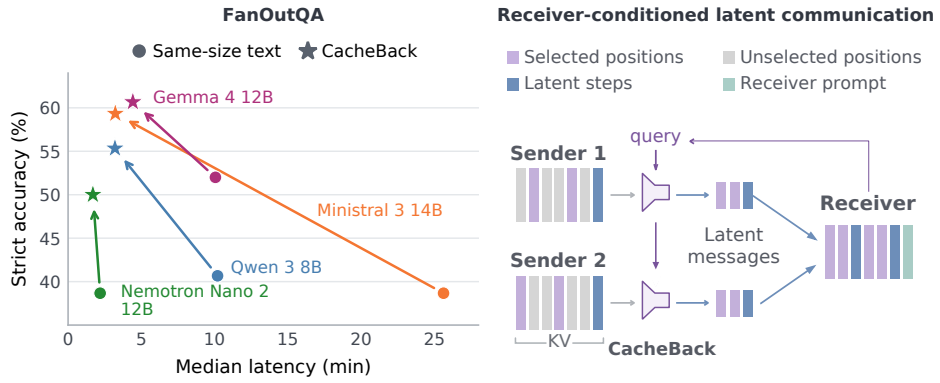

Figure 1: Left: highest-accuracy CacheBack setting per family versus same-size text on FanOutQA; $n = 50$ concurrent tasks on $8\times$ H100. Right: the receiver query guides selection of sender KV states.

1 INTRODUCTION

Agents commonly distribute information and work across models to exploit parallelism, manage context limits, and divide tasks into specialized steps (Anthropic, 2025; Zhang et al., 2024; Fourney et al., 2024). We consider directed acyclic graphs, where nodes (agents) send messages from sender to receiver along directed edges; an interior node may both send and receive messages. Common patterns include fan-in, where work is partitioned across multiple agents who send their outputs to a single receiver, and a sequential chain, where agents process incoming messages and send to the next.

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.

For example, a FanOutQA task asks, "*What is the highest elevation (in feet) of the five largest U.S. states by land mass?*" (Zhu et al., 2024). Answering it requires identifying the five states from one Wikipedia article, then finding each state's highest elevation in five additional articles. With four agents, we may give two articles to three of the agents to read, and they send summaries to a receiving agent that combines and answers with Alaska 20,310; Texas 8,751; California 14,505; Montana 12,807; New Mexico 13,161.

While text is the dominant form of communication, discrete tokens are less expressive than continuous hidden representations and are costly to generate (Zou et al., 2026). In our FanOutQA evaluation, generating three Qwen 3 8B text messages takes a median 73.2 seconds.[1] Systems manage accuracy–latency trade-offs through shorter messages (Chen et al., 2025) or delegation to smaller models (Feng et al., 2026).

Recent work proposes to use the KV cache for agent communication. LatentMAS generates latent thoughts by autoregressing over hidden representations rather than decoded tokens: each latent step feeds its final hidden representation back as the next continuous input. After G latent steps, an agent sends its full KV cache, containing its input context and generated latent steps, as the message. Assuming that the receiving agent uses the same model, it prepends this message to its KV cache and immediately continues computing. Across reasoning and code benchmarks, LatentMAS improves average accuracy by 2.8–4.6 percentage points and end-to-end latency by $4.0\times$ in sequential and $4.3\times$ in hierarchical topologies over text communication (Zou et al., 2026).

However, communicating full KV caches does not scale to long contexts. When subagents read different articles, their KV caches differ and must all be sent to the receiver. Thus, the receiver's input context, the amount of transferred data, the memory pressure, and receiver attention must all grow linearly with the number of subagents and article sizes. This removes the context-management benefit of partitioning evidence across agents. In our FanOutQA settings with Qwen 3 8B, full-cache transfer leaves insufficient context for receiver generation on every task (Section 4.2).

We observe that the full KV cache also transmits information the receiver does not need, and that **what an agent sends should depend on the receiver's information need**. We argue for a *receiver-conditioned communication* protocol, where the receiver shares its information need with its subagents, who use the need to selectively choose the information to send. We instantiate this idea in **CacheBack**, a simple, robust, and training-free selector for communication between the same models that can reduce the message size from $2\times$ to over $100\times$.

When compared to text communication between same-size models, CacheBack improves strict accuracy by 7.3–20.7 percentage points and reduces task-completion latency by $1.3\times$–$8.0\times$ on FanOutQA, and improves accuracy by 9.3–14.7 percentage points with $2.5\times$–$3.7\times$ speedups on LongBench v2. Across these benchmarks and four model families spanning dense Transformers, Mamba-attention hybrids, and sliding-window attention, we consistently shift the accuracy–latency Pareto frontier towards faster, more accurate task completions.

To summarize, we make three contributions. First, we introduce **receiver-conditioned communication**, where what an agent sends depends on the receiver's task needs. Second, we introduce **CacheBack**, a simple, robust, training-free method for receiver-conditioned latent communication that uses the receiver's query to select positions from the sender's sequence. Third, we show that receiver-conditioned latent communication outperforms same-size text communication when both use the same receiver query, in both accuracy and latency across model families and agent topologies.

2 Agent Communication

In a DAG agent topology, a message is sent along an edge from a sender agent i to a receiver agent j. Agent i first processes H_i input tokens, which include its source context and task instructions, and then produces a message for the receiver. Under text communication, that message contains K_i generated tokens. The sender decodes those tokens one at a time, and the receiver prefills them as token identifiers.

[1] Measured over the same 50 questions on one NVIDIA H100 80GB. Timing includes agent prefill and text generation but excludes receiver computation and transfer.

Latent communication instead sends internal model state. Prior work varies both how that state is produced and what form of it is transmitted. Coconut reasons through continuous hidden states (Hao et al., 2025), while LatentMAS generates autoregressive latent steps (Zou et al., 2026), which we keep fixed. Communication methods send embeddings, hidden states or activations, or KV state (Pham et al., 2024; Ramesh & Li, 2025; Du et al., 2026; Zheng et al., 2025; Zou et al., 2026; Fu et al., 2026; Shi et al., 2026; Liu, 2026). Some also learn mappings between sender and receiver representations or change how that state is fused at the receiver (Fu et al., 2026; Du et al., 2026; Kriuk & Ng, 2025; Rossi et al., 2026). These methods focus on the representation and transmission of latent state. We study which information from the completed sender state should enter the receiver's message.

For the KV-based communication we study, the sender processes H_i input tokens and then generates G_i latent steps. Its completed sequence therefore contains $T_i = H_i + G_i$ positions, and the message is drawn from this sequence. We call each entry of the sender sequence a position; the sender holds one KV entry per layer for each position. A position from an input token has a token ID, while a position from a latent step has no token ID and exists only as a continuous vector. A selected position is one included in the message. Its representation is how that position is encoded for transmission, either as a token ID or as one row of a continuous representation. In long-context settings where senders process large documents, $K_i \ll T_i$.

LatentMAS transmits KV state for all T_i positions at every layer (Zou et al., 2026). KVComm reduces the transmitted state by sharing only selected layers, but retains all T_i positions within those layers (Shi et al., 2026). Both therefore fix the message content to the sender's entire sequence while changing how much state is sent for each position.

2.1 Communication Costs

Full-cache communication sends the KV state for every sender-sequence position at every communicated layer; this costs 144 KiB per position for Qwen 3 8B in bfloat16.

Compression and quantization reduce the message size but not the number of positions the receiver must load. Even with free transfer costs, the receiver loads all T_i positions from each agent, in addition to the H_R positions in the receiver's own prompt. Thus, M subagents would consume a receiver's context size of:

$$H_R^{\text{text}} = H_R + \sum_{i=1}^{M} K_i, \qquad H_R^{\text{full}} = H_R + \sum_{i=1}^{M} T_i. \tag{1}$$

We call this context re-expansion. Although partitioning a long input (i.e., many documents) across subagents reduces the context processed per subagent, sending their full states still transfers considerable data over the network, accumulates the contexts at the receiver, consumes the receiver's KV memory, and degrades attention's effectiveness. In fact, our experiments find that the receiver often does not have enough context for its own prompt. We illustrate this in Figure 2.

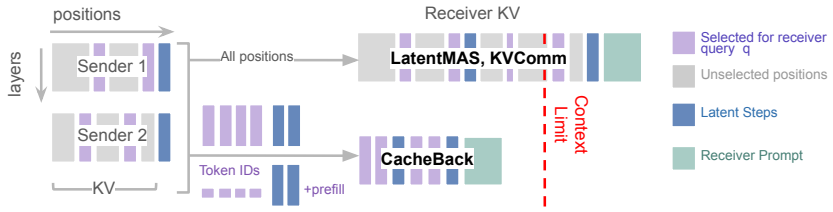

Figure 2: Receiver-conditioned communication limits context accumulation. Full-cache messages accumulate all agents' positions at the receiver. Receiver-conditioned messages retain selected source and latent positions, represented as token IDs and continuous inputs for receiver prefill.

2.2 The Need for Receiver-Conditioned Selection

At minimum, context re-expansion must leave room for the receiver's prompt and output. Let B_i bound agent i's selected positions, O_R reserve output positions, and L_R denote the context limit:

$$H_R + \sum_{i=1}^{M} B_i + O_R \leq L_R. \tag{2}$$

Here, B_i bounds positions after re-expansion, and H_R denotes the receiver's prompt length. A position budget limits how much an agent can send, but not which positions to send or how to represent them. Section 3 describes how the receiver states what it needs and how the sender uses that query; Appendix A formalizes these decisions and their quality–cost objective.

3 Receiver-Conditioned Communication

Receiver-conditioned communication has three parts. First, a receiver query q states what information the receiver needs. Second, the sender uses q to choose what information to send. Third, the selected information is represented for transmission to the receiver. Figure 2 shows how this reduces the positions sent to the receiver, and the message itself can be further compressed by sending token IDs and continuous vectors (where they do not correspond to vocabulary tokens) rather than KV contents.

The query q states what the receiver wants from the sending agent. It is a separate input from the task that prefills the sender's KV, so the sender can service queries from different receivers. In our experiments, q is the task question. We append it after the sender completes its original prefill and only use q to score sender positions.

3.1 Selecting State for the Receiver

Receiver conditioning does not prescribe how q should determine message content. While methods could generate new content, transform existing state, or select existing positions, we focus on a simple, training-free setting that selects $\leq B_i$ positions using the receiver's query. This design allows us to study the extent to which receiver conditioning helps.

Fortunately, KV-cache compression already provides methods for using a query to decide which source positions to retain, which receiver-conditioned communication sets to the receiver's query q. We study two selector variants: mean query attention, a naive selector based on existing KV-cache compression, and CacheBack, which changes how the query tokens are aggregated.

3.1.1 Naive selector

We build on SnapKV (Li et al., 2024), which scores each earlier position by the attention it receives from the final prompt tokens and keeps the highest-scoring state for the model's own continuation. We put the receiver's query q in those final positions instead: the sender appends q and only computes the query tokens against its existing KV cache.

Mean query attention averages the attention these query tokens pay to earlier positions across query tokens and query heads, sums across eligible layers, and pools nearby positions so that they receive similar scores, giving one score Q_t for each sender-sequence position. Because q is applied after the sender state is formed, changing the query changes the ranking, and the query can specify a different information need from the task that produced that state. Appendix C.1 describes aggregation, pooling, and eligible layers in more detail.

A limitation of averaging is that positions relevant to one part of the query may only receive strong attention from the few query tokens that refer to them. Thus, averaging over the whole query dilutes the signal. In Figure 3, position A receives strong attention from one query token, while B receives weaker attention from every token, so mean query attention ranks B above A even though A is the position the query asks for. A selector should not penalize a position for being relevant to only part of the query.

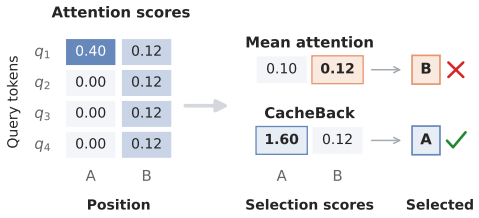

Figure 3: Averaging can hide query-specific support for A and choose irrelevant token B.

3.1.2 CACHEBACK

CacheBack proposes a simple correction to the averaging problem by reweighting the mean query attention score Q_t to give additional weight to positions whose attention is concentrated in particular query tokens: $S_t^{\mathrm{CB}} = Q_t C_t^2$.

Here C_t compares pooled attention before and after averaging query tokens. The correction is larger when a sender position receives concentrated support from part of the query. Appendix C.1 gives the exact definitions and pooling order.

We intentionally keep this correction simple and training-free to test whether using q improves communication without introducing a learned module.

State Representation. After scoring, we select positions under the message budget B_i[2]. We transmit each selected position and latent step as a continuous row (**continuous-row encoding**), which is about $18\times$ smaller per position than full KV state for Qwen3-8B. Source positions can be compressed further by sending their token IDs (**token-ID encoding**), while latent steps remain continuous vectors because they have no token IDs. The receiver prefills the token IDs, appends the generated KV positions, and then continues the task. Across 50 tasks, the token-ID encoding reduces payload by a further $75\times$ relative to continuous rows. While this is considerable over wide-area networks (e.g., $664 \rightarrow 9$ ms on a 300 Mbit/s Wi-Fi link), the $55 \rightarrow 0.7\,\mu$s difference on the 450 GB/s NVLink used in our evaluation is dwarfed by other costs (Figure 6). We therefore use continuous rows for all reported experiments; Appendix C.3 gives the full payload and transfer comparison.

3.2 SELECTION BEHAVIOR

Before the end-to-end evaluation, we use a small held-out set of 13 FanOutQA tasks as a diagnostic to check whether the receiver's query helps identify the evidence the receiver needs, and whether that depends on model size. Each task contains around 50K tokens of context, and we vary Qwen 3 sizes of 1.7B, 4B, 8B and 32B. We compare selectors that are receiver-independent[3] with the two receiver-conditioned selectors, mean query attention and CacheBack. For each selector and position budget, we use the task's annotated evidence to measure evidence recall in the selected positions. Appendix D reports the baseline adaptations and preliminary settings.

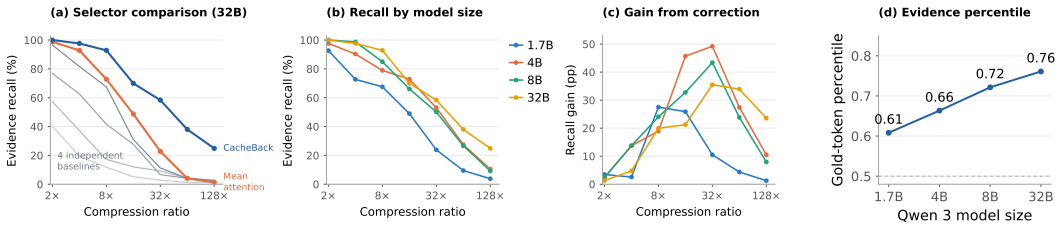

Figure 4: Results from diagnostic tests. (a) CacheBack retains high evidence recall even at $16\times$ compression compared to mean query attention and receiver-independent baselines. (b) CacheBack generally retains more evidence as model size increases. (c) The largest gains from CacheBack over mean query attention shift toward higher compression ratios with model size. (d) Mean query attention ranks gold tokens higher as model size increases.

Varying selectors. Figure 4(a) shows that as we reduce the number of positions by 2–$32\times$, mean query attention retains more evidence than every receiver-independent selector. At $8\times$, the strongest receiver-independent selector retains 68% of the evidence, compared with 93% for CacheBack. At $128\times$, CacheBack achieves 25% evidence recall, compared with 1–3% for the other selectors. We thus study CacheBack selection in our end-to-end experiments.

[2]The exact span-selection rule is given in Appendix C.2.

[3]We adapt StreamingLLM (Xiao et al., 2024), H_2O (Zhang et al., 2023), ChunkKV (Liu et al., 2025), and KVzip (Kim et al., 2025) for message selection.

Model scale. Figure 4(d) shows that mean query attention from the receiver query ranks gold tokens higher as model size increases. From Qwen 3 1.7B to 32B, the median gold-token percentile rises from 0.61 to 0.76, with 0.75 corresponding to the top 25% of positions.

CacheBack evidence recall also tends to improve with model scale. At $8\times$ compression, recall rises from 68% for 1.7B to 93% for 32B. Figure 4(c) shows another effect of scale, with CacheBack's largest gain over mean query attention shifting toward higher compression ratios in larger models.

These diagnostics show that the receiver query provides a useful signal for deciding what to send. We next test whether receiver-conditioned latent communication can improve both answer quality and latency over text communication across models and topologies.

4 End-to-End Evaluation

We test whether latent communication improves answer quality and latency over text when both use the receiver query. Text workers use the query to generate a report, while CacheBack uses it to select sender state. The comparison therefore changes how the message is constructed, not the information need used to construct it. Where the receiver context window permits it, we also evaluate a full rows control that sends every position without selection, using CacheBack's representation. At $16\times$ compression, CacheBack removes 94% of sender positions and has lower latency and higher accuracy than same text agents across every tested model family and communication topology (benchmark).

4.1 Experimental Setup

Our FanOutQA setting (Zhu et al., 2024) tests a parallel fan-in: three agents read different Wikipedia pages, then send messages to one receiver to answer the question. For LongBench v2 (Bai et al., 2025), we construct a sequential chain in which each agent reads the next part of a long document with the preceding agent's message.

We evaluate Qwen 3 (1.7B/4B/8B) (Yang et al., 2025), Ministral 3 (3B/8B/14B) (Liu et al., 2026), Gemma 4 (E2B/E4B/12B) (Gemma Team, 2026), and Nemotron Nano 2 (9B/12B) and 3 Nano (4B) (NVIDIA, 2025; 2026). FanOutQA uses all four families, while LongBench v2 uses Qwen and Nemotron, covering a dense Transformer and a hybrid architecture. Within each family, the receiver is fixed to the largest model. CacheBack also uses the largest model as sender and varies from $2\times$ to $128\times$ compression on FanOutQA and from $4\times$ to $128\times$ on LongBench v2. Text baselines use either the same model, which we call **same-size text**, or a smaller sender. We compare only within each family. For hybrid architectures, CacheBack scores positions using only their global-attention layers.

Every configuration runs on an $8\times$ 80 GB H100 GPU node. We choose the sender–receiver GPU allocation separately for each communication channel and keep that allocation fixed within the configuration. Models use their native chat templates, thinking modes, and recommended decoding settings. Text message lengths are not set from the latent message budgets.

For each benchmark, we evaluate on 50 tasks that are submitted as a batch. We report p50 and p95 time to end of answer (TTEOA), including queueing, sender computation, message preparation and transfer, receiver prefill, and receiver generation. Appendix F reports the serving settings, and Appendix G gives the prompts and input construction.

Figure 5 summarizes the Qwen and Nemotron accuracy–latency tradeoffs on both benchmarks. Appendix E.2 shows the remaining FanOutQA curves and Appendix E.4 the full LongBench results.

4.2 FanOutQA: Parallel Communication

This experiment tests a fan-in topology where one receiver aggregates messages from three senders. We sample 50 FanOutQA tasks from the official dev split, excluding the questions used for selector development. We keep only questions whose pages provide enough text for the split below, then take the first 50 by ascending SHA-256 hash of the question ID so no task is hand-picked. For each task, we partition the source pages among three senders, with at least 40K source tokens per sender and 120,000 in total. The receiver answers from their messages. We report strict accuracy, which requires every reference-answer group to appear in the answer. Appendix F.2 gives the question-selection and scoring details.

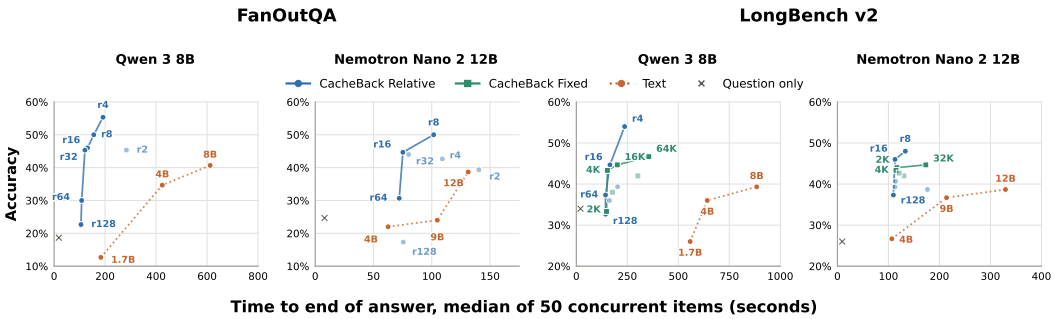

Figure 5: CacheBack improves accuracy–latency tradeoffs on FanOutQA (strict accuracy) and LongBench v2 (answer accuracy). rX: X-fold compression; xK: fixed budget of x thousand positions. Faded points are off-channel frontiers; LongBench labels mark frontier points only.

Table 1 shows that CacheBack improves strict accuracy by 7.3–20.7 percentage points and reduces median TTEOA by $1.3\times$–$8.0\times$ relative to same-size text. For Qwen and Ministral, CacheBack $32\times$ outperforms all tested text settings on both latency and accuracy. For Gemma and Nemotron, smaller text models are faster but sacrifice considerable accuracy. Figure 5 shows the Qwen and Nemotron curves; Figure 11 in Appendix E.2 includes all four families.

Table 1: Selected FanOutQA operating points versus same-size text. CB: CacheBack.

	Strict accuracy (%)			Median TTEOA (s)			Message recall (%)		
Model (ratio)	Text	CB	Gain (pp)	Text	CB	Speedup	Text	CB	Gain (pp)
Qwen 3 8B ($4\times$)	40.7	55.3	**+14.7**	612	192	**3.2×**	89.5	95.8	**+6.3**
Ministral 3 14B ($32\times$)	38.7	59.3	**+20.7**	1,537	193	**8.0×**	87.5	91.5	**+4.0**
Gemma 4 12B ($4\times$)	52.0	59.3	**+7.3**	603	201	**3.0×**	91.1	95.8	**+4.7**
Nemotron Nano 2 12B ($8\times$)	38.7	50.0	**+11.3**	131	102	**1.3×**	84.8	95.6	**+10.8**

Text messages are compact, but each additional token requires another autoregressive decoding step. CacheBack selects positions from state the agent has already computed, so it can transmit representations of more positions without generating them one at a time. At the selected operating points, these larger messages remain far smaller than the complete state and achieve 91.5%–95.8% message recall, compared with 84.8%–91.1% for same-size text (Table 1). The additional positions increase transfer and receiver prefill, but text spends more time generating its message and incurs more sender queueing under the concurrent workload (Figure 6).

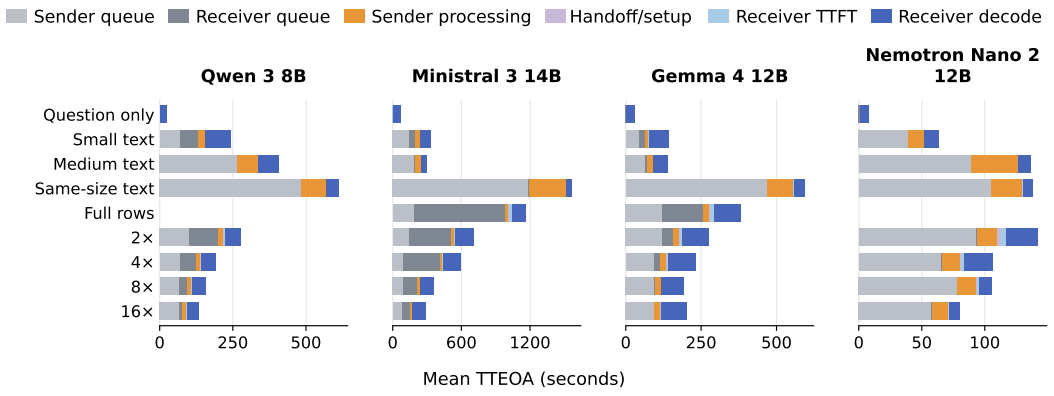

Figure 6: FanOutQA mean TTEOA by stage: queueing, agent processing, message preparation and transfer, receiver prefill, and generation. Breakdowns beyond $16\times$ are omitted as they change little.

We next isolate the role of the receiver query. We replace it with a fixed unrelated query, keeping sender computation, the message budget, and the final task question unchanged. Message recall falls at every compression ratio. At $16\times$, message recall falls from 91% to 65%, while strict accuracy falls from 46% to 23% (Figure 7). The receiver query therefore changes which evidence is retained and, in turn, final accuracy.

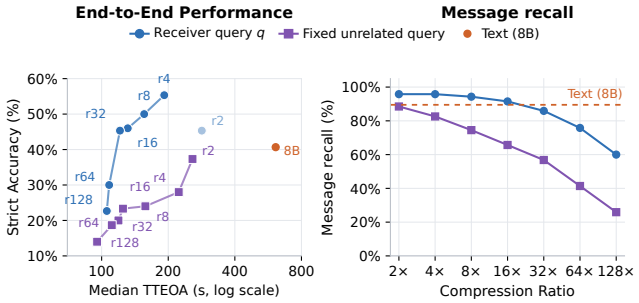

Figure 7: Effect of the receiver query q on FanOutQA for Qwen 3 8B. The unrelated query is "Why was Le Chaton Fat denied a small-business loan?"

Beyond the selected operating points, further compression yields little latency benefit and can sharply reduce accuracy (Figure 5, FanOutQA). For Ministral and Gemma, all sender positions fit in the receiver context, so we also send every position using the same representation as CacheBack. Comparing this with CacheBack isolates the effect of selection. Appendix E.2 reports the full results.

4.3 LongBench v2: Sequential Communication

We construct a sequential chain to test communication when information must survive multiple agent interactions. We split each 100K–246K-token LongBench v2 document into four equal parts of 25K–61K tokens. Four agents read these parts in order. Each agent receives the previous message, reads its own part, and selects positions from the combined state to form the next message. A final receiver answers from the fourth message. Information from early parts must therefore survive several rounds of message construction before reaching the receiver.

We evaluate this chain on 50 LongBench v2 questions. In an initial run, many text sender messages hit generation limits. For simplicity, we restrict the final panel to the Easy subset as one of several changes that reduce message truncation. This preserves a fair comparison between text and latent communication. Appendix F.3 gives the full evaluation changes. Since each message is appended to the receiver's context, its size grows linearly with chain length and the context added at each step. If each agent adds H positions, the message grows to $O(HN)$ after N steps (Section 2.1).

To control this growth, we evaluate two message budgets. Under the relative budget with compression ratio r, the budget increases at each agent by $1/r$ of the positions that agent adds. If each agent adds H positions, the message budget therefore grows by H/r per agent and reaches NH/r after N agents. Under the fixed budget B, the outgoing message contains at most B positions, so its size stops growing once B is reached. Under both budgets, each agent reranks inherited and newly added positions together, then selects up to the budget. Inherited positions can therefore be dropped from later messages. Appendix G.2 gives the prompts and input construction.

The LongBench panels in Figure 5 show that CacheBack shifts the accuracy–latency frontier, as on FanOutQA. For Qwen, Relative at $4\times$ and $16\times$ achieves higher accuracy and lower latency than every text baseline. For Nemotron, a smaller text agent remains faster but less accurate. Figure 12 in Appendix E.4 provides the fully labeled curves.

Table 2: Selected operating points on LongBench v2 Easy versus same-size text. CB: CacheBack.

		Accuracy (%)			Median TTEOA (s)		
Model	Setting	Text	CB	Gain (pp)	Text	CB	Speedup
Qwen 3 8B	Relative $4\times$	39.3	54.0	**+14.7**	884	237	**3.7×**
	Fixed 64K	39.3	46.7	**+7.3**	884	355	**2.5×**
Nemotron Nano 2 12B	Relative $8\times$	38.7	48.0	**+9.3**	329	133	**2.5×**
	Fixed 32K	38.7	44.7	**+6.0**	329	173	**1.9×**

Table 2 shows that **Relative** CacheBack improves accuracy by 14.7 points for Qwen and 9.3 for Nemotron, with $3.7\times$ and $2.5\times$ lower median TTEOA. **Fixed** CacheBack improves accuracy by

7.3 and 6.0 points, with $2.5\times$ and $1.9\times$ lower median TTEOA. In the sequential chain, each agent must finish its message before the next can begin. Text therefore pays autoregressive decoding at all four steps, increasing both sender processing and queueing. CacheBack avoids this decoding cost (Appendix E.5).

More aggressive Relative compression yields diminishing latency benefits and degrades accuracy. For Qwen, increasing the compression ratio from $16\times$ to $128\times$ reduces median TTEOA from 165 to 143 seconds, but lowers accuracy from 44.7% to 32.7%.

5 Discussion and Limitations

Receiver-conditioned latent communication retains the context-management benefit of delegation without autoregressive text generation. The sender constructs its message by selecting positions from state it has already computed. While sending more selected positions increases transfer volume, receiver memory use, and prefill work, selecting them requires no additional sender decoding. The sender can therefore send more of its computed state without decoding a longer text message.

Avoiding sender decoding saves the most time when agents would otherwise spend a large share of task time decoding text messages. Long or repeated messages create this condition when they delay dependent work or keep sender GPUs busy under concurrent load. Repository exploration is one example, where several agents may need to return detailed findings before planning can continue. By contrast, an editing agent may spend most of its time changing files, running tests, or using tools before returning a short message. The advantage narrows when network transfer or receiver prefill costs more than the sender decoding avoided.

Because q is applied after sender computation, the same completed state can be scored for different receiver queries. Each new query requires another selection, but not rerunning the sender's task or decoding a separate text message. One sender computation could therefore support several receiver-specific messages or follow-up queries from the same receiver. We do not evaluate how many such queries a sender can serve before selection and transfer become limiting.

Our primary goal is to isolate whether conditioning communication on the receiver's information need can limit context re-expansion while preserving the benefits of latent communication. We therefore set the receiver query to the task question, issue it once after sender computation, and fix the message budget before execution. Across our long-context QA workloads, the receiver query changes which information is selected and improves the quality–latency tradeoff. This isolates the effect of receiver conditioning, but does not determine how queries should be written, when a receiver should ask again, or how much state it should request. A receiver could instead ask for evidence about an unresolved claim, request a specific file or dependency, or ask for more detail after an initial message. The query and message budget could then change as the receiver's task changes.

We make the same choice at the level of the selector. CacheBack is deliberately simple and training-free so that the experiment tests receiver conditioning without requiring a learned communication module. It uses a fixed attention-based scoring rule, and hybrid models expose only their global-attention layers to that rule. Despite these restrictions, receiver conditioning shifts the quality–latency frontier across every tested model family and topology. These experiments do not establish the best selector. Learning the scoring rule or using more of the model state may improve selection further.

6 Conclusion

The emergence of latent communication has shown that changing message representation can avoid text generation and improve accuracy and latency. While these gains come from representation, representation alone does not determine what information should be sent. Full-cache latent methods still send every position in the sender sequence, so the receiver can accumulate the context that was partitioned across agents. Receiver-conditioned communication instead separates these choices. CacheBack shows that even a simple, training-free method can shift the quality–latency frontier across model families and communication topologies. A communication protocol should therefore decide not only how a message is represented, but what information it contains. And what information it contains should depend on what the receiving agent needs.

ETHICS STATEMENT

This work involves no human subjects or crowd workers. All experiments draw on two public benchmarks: FanOutQA, which is built from English Wikipedia, and LongBench v2, whose documents were collected from public sources and whose questions were written by the benchmark's own annotators. We redistribute prepared FanOutQA inputs and provide scripts to reconstruct the LongBench v2 inputs. We are aware of no fairness or bias concerns specific to this work beyond those already present in the underlying models and benchmarks. Two foreseeable risks arise from compressing communication between agents. First, a compressed handoff can discard detail that the receiver needed, which can lead to downstream errors. Second, a latent message offers no confidentiality guarantee, because it is derived from the worker's private context. Latent messages are not human-readable. We follow LatentMAS's latent protocol, whose debug mode generates parallel text probes for inspection (Zou et al., 2026). The authors declare no competing interests.

REPRODUCIBILITY STATEMENT

Appendix C defines the selector and how each message is constructed, including the position budget at every compression ratio. Appendices F and G give the prompts issued to workers and receivers, the benchmark preparation, the settings for every reported run (model checkpoints, decoding parameters, chunking, and seeds), the scoring procedure, and the serving configuration of the eight-GPU node on which latency was measured. We include with the submission, and will release publicly upon acceptance, our evaluation code and configurations for FanOutQA and LongBench v2, prepared FanOutQA inputs, and scripts to reconstruct the LongBench v2 inputs. Reruns may not match our numbers exactly. Sampled decoding and batch-dependent numerics in vLLM under concurrent load can change individual outputs even with fixed seeds, and latency depends on hardware, interconnect, vLLM version, and scheduling. Appendix E reports receiver sampling variation across three draws.

AI USE STATEMENT

The research idea, method, and experimental design are the authors' own; this is not automatically generated research. AI assistants were used, under the authors' direction, for implementation, running experiments, preparing figures and tables, surveying related work, and editing the manuscript. All code and pipelines were reviewed by the authors, every reported number traces to a result artifact the authors inspected, and the authors take full responsibility for the final work.

REFERENCES

Anthropic. How we built our multi-agent research system. Anthropic Engineering, 2025. URL https://www.anthropic.com/engineering/multi-agent-research-system.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, 2025. URL https://aclanthology.org/2025.acl-long.183/.

Weize Chen, Jiarui Yuan, Chen Qian, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Optima: Optimizing effectiveness and efficiency for LLM-based multi-agent system. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 11534–11557. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-acl.601. URL https://aclanthology.org/2025.findings-acl.601/.

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 64th Annual Meeting of the Association for Computational Linguistics*, 2026. URL https://aclanthology.org/2026.acl-long.1248/.

Tao Feng, Haozhen Zhang, Zijie Lei, Peixuan Han, and Jiaxuan You. GraphPlanner: Graph memory-augmented agentic routing for multi-agent LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2604.23626.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. Magentic-one: A generalist multi-agent system for solving complex tasks, 2024. URL https://arxiv.org/abs/2411.04468.

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*, 2026. URL https://arxiv.org/abs/2510.03215.

Gemma Team. Gemma 4 technical report, 2026. URL https://arxiv.org/abs/2607.02770.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason E. Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Itxz7S4Ip3.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. KVzip: Query-agnostic KV cache compression with context reconstruction. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/f4eaa4b8f2d08edb3f0af990d56134ea-Abstract-Conference.html.

Boris Kriuk and Logic Ng. Q-KVComm: Efficient multi-agent communication via adaptive KV cache compression, 2025. URL https://arxiv.org/abs/2512.17914.

Yiping Li, Zhiyu An, and Wan Du. When less latent leads to better relay: Information-preserving compression for latent multi-agent LLM collaboration, 2026. URL https://arxiv.org/abs/2604.13349.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/28ab418242603e0f7323e54185d19bde-Abstract-Conference.html.

Alexander H. Liu, Kartik Khandelwal, et al. Ministral 3, 2026. URL https://arxiv.org/abs/2601.08584.

Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Liuyue, Bo Li, Xuming Hu, and Xiaowen Chu. ChunkKV: Semantic-preserving KV cache compression for efficient long-context LLM inference. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/2987f911151b39cd3a1761e212319e8e-Abstract-Conference.html.

Yingzhuo Liu. Beyond tokens: A unified framework for latent communication in LLM-based multi-agent systems, 2026. URL https://arxiv.org/abs/2606.05711.

NVIDIA. NVIDIA Nemotron Nano 2: An accurate and efficient hybrid mamba-transformer reasoning model, 2025. URL https://arxiv.org/abs/2508.14444.

NVIDIA. NVIDIA Nemotron 3 Nano 4B BF16. Hugging Face model card, 2026. URL https://huggingface.co/nvidia/NVIDIA-Nemotron-3-Nano-4B-BF16. Accessed: 2026-09-23.

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*, 2024. URL https://openreview.net/forum?id=sehRvaIPQQ.

Vignav Ramesh and Kenneth Li. Communicating activations between language model agents. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 51094–51116. PMLR, 2025. URL https://proceedings.mlr.press/v267/ramesh25a.html.

Maximillian Rossi, Prajwal Raghunath, and Eugene Wu. Latent cache flow: Model-to-model communication without text, 2026. URL https://arxiv.org/abs/2605.22863.

Xiangyu Shi, Marco Chiesa, Gerald Q. Maguire, Jr., and Dejan Kostić. KVComm: Enabling efficient LLM communication through selective KV sharing. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=F7rUng23nw.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/5e5fd18f863cbe6d8ae392a93fd271c9-Abstract-Conference.html.

An Yang, Anfeng Li, et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Ö. Arık. Chain of Agents: Large language models collaborating on long-context tasks. In *Advances in Neural Information Processing Systems*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/ee71a4b14ec26710b39ee6be113d7750-Abstract-Conference.html.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/6ceefa7b15572587b78ecfcebb2827f8-Paper-Conference.pdf.

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*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/b2b502c3629beadda06311386d2c6f73-Abstract-Conference.html.

Andrew Zhu, Alyssa Hwang, Liam Dugan, and Chris Callison-Burch. FanOutQA: A multi-hop, multi-document question answering benchmark for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://aclanthology.org/2024.acl-short.2/.

Jiaru Zou, Ruizhong Qiu, Gaotang Li, Xiyuan Yang, 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 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2511.20639.

APPENDIX: TABLE OF CONTENTS

A Formal Model of Receiver-Conditioned Communication **15**

B Comparison with OBF **15**

C Selector and message definitions **16**

C.1 General scoring rule

C.2 Budgets and position selection 17

C.3 Message encoding

D Selector development **18**

D.1 Evidence recall

D.2 Parameter sensitivity 19

E Results and receiver sampling **20**

E.1 FanOutQA message recall

E.2 Full FanOutQA results 21

E.3 FanOutQA receiver sampling 22

E.4 LongBench 23

E.5 LongBench latency breakdown

F Serving and evaluation settings **24**

F.1 Timing and output 25

F.2 FanOutQA evaluation and scoring

F.3 LongBench evaluation and scoring 26

G Prompts and input construction **27**

G.1 FanOutQA prompts

G.2 LongBench prompts 28

G.3 Chat-template packing 29

A Formal Model of Receiver-Conditioned Communication

We formalize the communication choices introduced in Section 2. A receiving agent provides a query q describing the information it seeks from a completed agent state. Message construction then separates two choices: what information is selected and how that information is represented for transmission.

Communication model. A sender S_θ and receiver R_θ share model weights θ (or a mapping between their representations) and cooperate on task T. The sender processes input x, producing a layer-wise KV cache, where j indexes positions in the sender sequence:

$$\mathcal{M}_S = \left\{ (K_j^{(\ell)}, V_j^{(\ell)}) \right\}_{\ell=1,\ldots,L;\ j=1,\ldots,N} \tag{3}$$

Let r denote the receiver's current context. The content map C_ϕ determines what information to communicate for the query q. The representation map P_ψ determines its transmitted form:

$$c = C_\phi(\mathcal{M}_S; q), \qquad \tau = P_\psi(c), \tag{4}$$

The receiver decodes the message and uses it to produce its output,

$$\hat{y} = R_\theta(T, r; D_\psi(\tau)). \tag{5}$$

Quality and cost. For task target y and loss ℓ, we measure message adequacy by the receiver's expected task loss,

$$\mathcal{E} = \mathbb{E}[\ell(\hat{y}, y)], \tag{6}$$

averaging over tasks and generation randomness. Let J be a measured cost under fixed models, hardware, and workload, such as median task-completion latency. This latency includes worker computation, message construction and transfer, receiver processing, and scheduling delays. Both cost and task loss depend on the content map, representation map, and position budgets $\mathbf{B} = (B_1, \ldots, B_M)$. Among compatible implementations satisfying Equation 2, we seek

$$\underbrace{\min_{\phi,\psi,\mathbf{B}} J}_{\text{minimize cost}} \qquad \text{subject to} \qquad \underbrace{\mathcal{E} \le \delta}_{\text{meet the quality target}}, \tag{7}$$

where δ is the maximum acceptable task loss. Varying δ traces the quality–cost frontier for the chosen models, workload, and design space.

A text report chooses content through autoregressive generation. When generated for the receiver's query q, it implements a receiver-conditioned content map. A report generated only for the worker's original assignment may omit information needed for a different receiver query. Receiver conditioning is therefore a property of message construction, not of whether the message is textual or latent.

LatentMAS instead fixes $C_\phi(\mathcal{M}_S; q) = \mathcal{M}_S$ (Zou et al., 2026). It returns the entire state rather than choosing information for the receiver query. Layer selection and input-embedding transfer can reduce the transmitted payload while retaining all source positions. When $B_i < T_i$, however, message construction must also choose which positions to omit. A choice made without q cannot adapt to different queries over the same worker state.

B Comparison with OBF

OBF studies compressed KV relay in a sequential reasoning chain whose agents receive the same question (Li et al., 2026). Our experiments instead distribute source evidence across workers: different articles in FanOutQA and successive document chunks in LongBench v2. The receiver must obtain information from those private inputs through the handoffs.

Table 3: Information access and context growth. Shared-question reasoning versus communication of private evidence.

	OBF evaluation	**Our evaluation**
Information	Same question at every agent; no private source documents	Agents receive distinct evidence unavailable to the receiver
Selection signal	Attention from current latent reasoning states	Attention from the receiver's query
Topology	Sequential reasoning chain	Sequential (LongBench v2) and parallel (FanOutQA)
Total input	496–1,012 prompt tokens across three reasoning agents	FanOutQA: 122k–150k evidence tokens; LongBench v2: 100k–246k source tokens
Context growth	Inherited state is preserved; retained prompt KV and new thoughts accumulate	Relative budgets grow with added context; fixed budgets cap the message size. Fan-in combines compressed agent messages

Input counts: OBF reports mean prompt tokens, including repeated prompts; ours count source tokens once per task using the Qwen tokenizer.

C Selector and message definitions

C.1 General scoring rule

Adapting SnapKV. We adapt SnapKV's attention-based scoring (Li et al., 2024), using the receiver's query in place of the sender's prompt suffix. The sender computes attention from the query q to its completed KV cache. Only attention from query tokens is used to compute the selection scores. Let m be the number of query tokens, $\mathcal{L}$ the set of attention layers used for scoring, and $\mathcal{H}_\ell$ the query heads in layer ℓ. Write $a_{\ell,h,r,t}$ for attention from query token r, in query head h of layer ℓ, to sender position t. If a model uses both local and global attention, we use only global-attention layers for scoring. We use Pool_7 to denote a local maximum over seven sender positions, centered on position t and clipped at sequence boundaries. Pooling is only over sender positions.

Mean query attention. For each layer in $\mathcal{L}$, we average attention over query tokens and query heads. We then sum across layers and pool over sender positions:

$$Q_t = \mathrm{Pool}_7\left(\sum_{\ell\in\mathcal{L}} \frac{1}{m|\mathcal{H}_\ell|} \sum_{r=1}^{m} \sum_{h\in\mathcal{H}_\ell} a_{\ell,h,r,t}\right). \tag{8}$$

This gives one shared ranking of sender positions across query heads.

CacheBack's correction. Averaging across query tokens can hide a sender position that receives strong attention from one part of the query. CacheBack compares attention before and after this averaging. We first average query heads that share each KV head. Let $\mathcal{S}$ be the set of these KV-head groups across layers in $\mathcal{L}$, and let $b_{s,r,t}$ be the mean attention in KV-head group s from query token r to sender position t. For $p > 1$, define

$$C_{p,t} = \left(\frac{\sum_{s\in\mathcal{S}} \mathrm{mean}_r\left[\mathrm{Pool}_7(b_{s,r,t})^p\right]}{\sum_{s\in\mathcal{S}}\left[\mathrm{Pool}_7(\mathrm{mean}_r\, b_{s,r,t})\right]^p}\right)^{1/(p-1)}, \qquad C_t \equiv C_{2,t}. \tag{9}$$

For the numerator, we pool over sender positions and raise to p before averaging over query tokens. For the denominator, we average over query tokens before pooling and taking the pth power. CacheBack uses $S_t^{\mathrm{CB}} = Q_t C_t^2$. Appendix D.2 varies p and how strongly $C_{p,t}$ changes the score.

Interpretation. At $p = 2$, without pooling, the correction reduces to one plus a normalized variance across query tokens:

$$C_t = 1 + \frac{\sum_s \mathrm{Var}_r(b_{s,r,t})}{\sum_s (\mathrm{mean}_r\, b_{s,r,t})^2}. \tag{10}$$

The identity holds when the denominator is nonzero; Var_r is the population variance over query tokens. With pooling, query tokens can peak at different positions within the same seven-position window, which can also increase C_t. When both sums are zero, we set $C_{p,t} = 1$.

C.2 Budgets and position selection

Mean query attention and CacheBack select positions using the same rule. We keep the first position and all 40 current latent positions, then add positions from ranked spans until the budget is full. These 41 positions count toward the budget. For FanOutQA, a sender i with T_i prompt and latent positions selects

$$B_i = \lceil T_i / r \rceil \tag{11}$$

positions at compression ratio r, so $B_i - 41$ positions remain for span selection. The count T_i includes prompt framing and the 40 latent positions. Every evaluated budget contains at least 41 positions.

In a sequential chain, let T_i^{old} count inherited positions and T_i^{new} count newly added prompt and latent positions. We evaluate two budget rules,

$$B_i^{\mathrm{relative}} = T_i^{\mathrm{old}} + \lceil T_i^{\mathrm{new}} / r \rceil, \qquad B_i^{\mathrm{fixed}} = \min(B_{\max}, T_i^{\mathrm{old}} + T_i^{\mathrm{new}}). \tag{12}$$

For B_i^{relative}, inherited positions increase the budget, but inherited and new positions are ranked together. We always keep the first position and the 40 current latent positions. Inherited latent positions can be discarded. Discarded positions are not available in later messages.

Spans begin at position zero, have width 16, and are ranked by their mean score. The last span may be shorter. Before ranking spans, we set the first position's score to the maximum position score. Span means include the first position, so its maximum score contributes to the first span's mean score. Equal span scores favor the earlier span.

We add each span's unselected positions in rank order until the budget is full. If the next span would exceed the budget, we add a shorter interval around its highest-scoring position. We return selected positions in source order.

C.3 Message encoding

Section 2 separates what is sent from how it is represented. Our experiments transmit every selected position as one continuous row and reconstruct the receiver KV cache with a fresh prefill. Because we re-prefill the selected rows, the receiver applies positional encodings at their new positions; direct KV transfer would require handling the sender–receiver positional mismatch explicitly, which we do not evaluate. The token-ID encoding instead sends selected source positions as token IDs and latent positions as continuous vectors. For the same model, the receiver reconstructs the corresponding source embeddings exactly from those token IDs. Figure 8 compares the resulting payloads and bandwidth-only transfer times. On the node-local links we evaluate, continuous-row transfer is small relative to other costs, so all reported experiments use it. Implementations that minimize transmitted bytes should use the token-ID encoding. Appendix F specifies scaling and alignment, and Appendix G specifies message placement and chat framing.

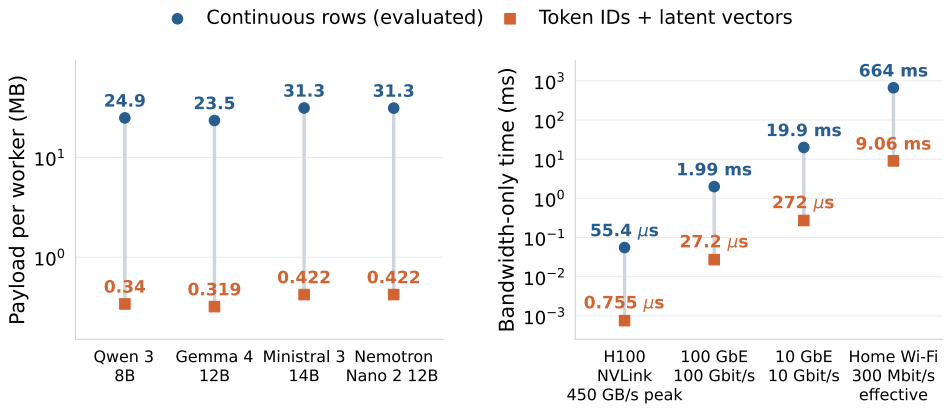

Figure 8: Continuous-row versus token-ID encoding with latent vectors. Left: payload per sender. Right: bandwidth-only transfer time. Both panels use log scales.

D SELECTOR DEVELOPMENT

D.1 EVIDENCE RECALL

We use 13 FanOutQA development questions for which every required piece of evidence not stated in the question can be located in the context provided to the agents. For each question, an evidence group counts as retained only when the selected positions keep every token of each required piece of evidence in that group. If the same evidence appears more than once, keeping any one copy is sufficient. Evidence recall averages the retained fraction across the 13 questions.

We compare StreamingLLM, H_2O, ChunkKV, KVzip, mean query attention, and CacheBack. StreamingLLM ranks positions by recency. H_2O sums causal prompt attention for each position. ChunkKV scores positions from the final 32 prompt rows, averages over a width-5 local window, and assigns positions the mean score of their 16-position chunk. KVzip scores sender positions using attention during context reconstruction. All six selectors rank sender positions and use the same position-selection rule and budget from Appendix C.2. We fixed width-16 spans and Pool_7 during preliminary development before this comparison rather than optimizing them on these 13 questions.

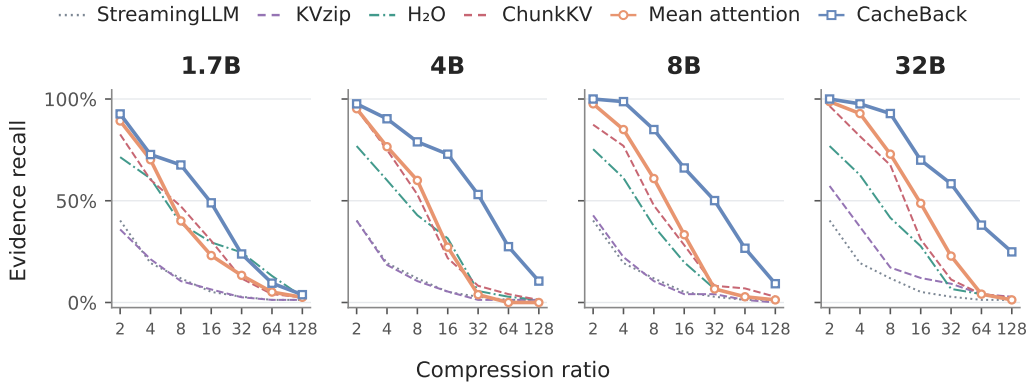

Figure 9: Mean evidence recall by selector across Qwen 3 models ($n = 13$).

Figure 9 compares evidence recall across Qwen 3 1.7B, 4B, 8B, and 32B. CacheBack has the highest recall at 26 of the 28 model-size–ratio settings. The two exceptions are Qwen 3 1.7B at 32× and 64×, where H_2O retains more evidence. We therefore use CacheBack in the end-to-end evaluation.

D.2 PARAMETER SENSITIVITY

We vary the power $p > 1$ inside the correction in Equation 9 and the strength $\alpha \geq 0$, which controls how strongly the correction changes Q_t. The score is

$$S_t(p, \alpha) = Q_t C_{p,t}^{\alpha}. \tag{13}$$

Mean query attention uses $\alpha = 0$. CacheBack uses $p = \alpha = 2$.

We selected $p = \alpha = 2$ on the 13 FanOutQA development questions and kept them fixed across model sizes, compression ratios, and the end-to-end evaluation.

At $p = \infty$, the correction compares the largest pooled query-token attention with the largest value after averaging query tokens.

Table 4: Evidence recall (%) over 13 tasks and four Qwen 3 sizes. Shading marks $p = \alpha = 2$; bold marks the best value at each ratio.

α	p	2×	4×	8×	16×	32×	64×	128×
Mean query attention								
0	–	95	81	58	33	12	3	1
CacheBack								
0.25	1.5	96	86	67	41	15	3	1
0.5		96	86	70	50	21	9	1
1		95	**90**	78	57	33	16	4
2		**98**	**90**	**81**	**65**	**46**	22	12
4		**98**	89	74	60	40	17	10
0.25	2	96	85	67	41	15	3	1
0.5		96	86	70	49	22	9	2
1		94	88	76	57	34	16	5
2	**2**	**98**	**90**	**81**	64	**46**	25	12
4		97	89	75	61	42	23	12
0.25	3	96	85	67	38	15	3	1
0.5		96	86	70	47	20	8	2
1		94	89	78	55	31	15	5
2		96	89	78	64	42	25	12
4		96	87	76	61	41	24	**13**
0.25	4	96	85	66	39	15	4	1
0.5		96	85	69	44	20	8	2
1		94	89	76	54	29	15	3
2		96	89	77	63	42	25	12
4		96	87	74	62	39	24	12
0.25	8	96	84	66	38	15	4	1
0.5		96	85	68	44	17	8	2
1		94	88	74	51	26	14	3
2		96	**90**	76	63	41	20	11
4		95	89	76	61	42	24	12
1	∞	94	88	71	51	25	13	3
2		96	89	77	61	40	20	9
4		95	89	77	61	41	**28**	11
8		96	86	70	59	40	26	11

E RESULTS AND RECEIVER SAMPLING

E.1 FANOUTQA MESSAGE RECALL

Our goal is to measure how much reference evidence reaches the receiver in the sender messages. We therefore compute message recall before receiver generation as the fraction of the 495 FanOutQA reference groups present in the three sender messages. For text, we match accepted answer strings in the generated messages. For CacheBack, we match accepted answer strings within retained source-token spans without joining across discarded positions, and exclude generated continuous inputs. Appendix F.2 gives the exact matching rules. Table 5 and Figure 10 report recall for every setting.

For CacheBack, this matching rule has a 96.4% ceiling. Eight reference groups appear in the question text and are included in the transmitted prompt framing. Another 18 do not appear in any sender context, so retained source tokens cannot contain them. The remaining 469 groups can be located in the sender context. Thus 477 of 495 groups can be matched. The full-row controls for Ministral and Gemma recover all 477 groups.

At 2×, CacheBack messages contain 474 to 477 of the 495 reference groups across the four model families. Recall remains above 91% through 16× in every family. It falls to 86.1–91.5% at 32×, 74.7–82.0% at 64×, and 50.3–62.8% at 128×.

Same-size text messages contain 84.8–91.1% of the reference groups across model families. At the CacheBack operating points in Table 1, the messages contain 91.5–95.8%. We pool message recall over reference groups, whereas strict accuracy requires every reference group for a question. A single missing group can therefore make the entire question strictly incorrect.

Table 5: FanOutQA message recall over 50 questions. CacheBack recall has a 96.4% ceiling.

Setting	Qwen 3	Ministral 3	Gemma 4	Nemotron
Text channels				
Small	72.1	82.6	86.7	74.5
Medium	86.7	90.5	87.1	77.6
Same-size	89.5	87.5	91.1	84.8
Latent channels (CacheBack)				
Full rows	–	96.4	96.4	–
2×	95.8	96.4	96.4	96.4
4×	95.8	95.6	95.8	96.2
8×	94.3	95.4	94.9	95.6
16×	91.5	94.5	92.9	93.5
32×	86.1	91.5	87.1	89.9
64×	76.0	82.0	74.7	78.8
128×	60.2	62.8	50.3	58.0

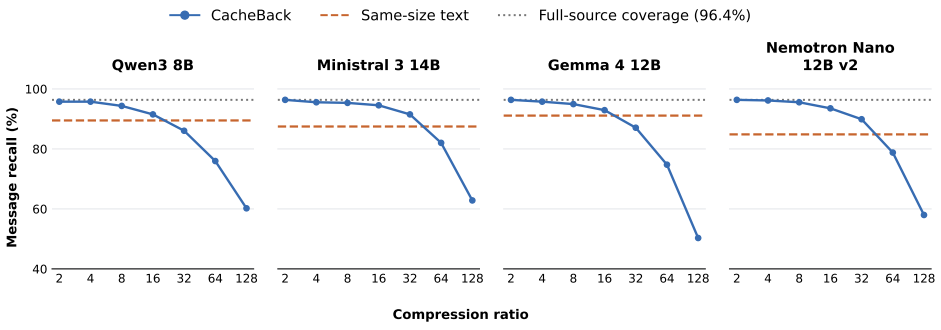

Figure 10: FanOutQA message recall over 50 questions and 495 reference groups.

E.2 FULL FANOUTQA RESULTS

Tables 6 and 7 report loose and strict accuracy, p50 and p95 TTEOA, and speedup over same-size text for every FanOutQA setting. TTEOA runs from question submission to the end of the first receiver answer. It includes queueing, sender computation, message preparation and transfer, receiver prefill, and receiver decoding. All 50 questions are submitted together, so TTEOA also includes queueing caused by concurrent questions on the same node.

Figure 11 shows strict accuracy against median TTEOA for all four model families.

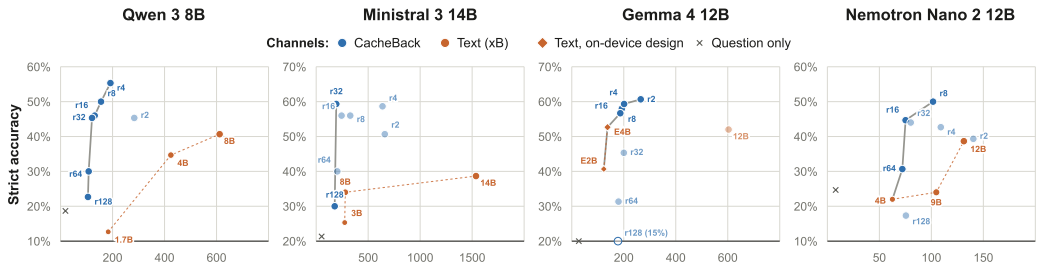

Figure 11: FanOutQA strict accuracy and median TTEOA for all evaluated settings. rX denotes CacheBack compression ratio X.

Table 6: Full FanOutQA results for global-attention Transformers. Loose and strict accuracy (%) over 50 questions and three receiver draws per setting.

	Qwen 3 8B receiver					**Ministral 3** 14B receiver				
	Accuracy ↑		**TTEOA**			**Accuracy ↑**		**TTEOA**		
Setting	Loose	Strict	p50 (s)	p95 (s)	Speedup	Loose	Strict	p50 (s)	p95 (s)	Speedup
Question only	57	19	19	62	–	53	21	53	309	–
Text channels										
Small	53	13	183	515	3.3×	63	25	274	709	5.6×
Medium	73	35	424	661	1.4×	69	34	280	552	5.5×
Same-size	78	41	612	1059	1.0×	74	39	1537	2750	1.0×
Latent channels										
Full rows	Exceeds context limit					72	44	1041	2296	1.5×
2×	83	45	284	503	2.2×	74	51	660	1278	2.3×
4×	**84**	**55**	192	322	3.2×	83	**59**	638	1048	2.4×
8×	**84**	50	156	264	3.9×	80	56	328	634	4.7×
16×	81	46	131	221	4.7×	81	56	244	669	6.3×
32×	82	45	121	201	5.1×	**84**	**59**	193	483	8.0×
64×	76	30	108	186	5.7×	80	40	203	**471**	7.6×
128×	70	23	**106**	**182**	**5.8**×	74	30	**178**	**471**	**8.6**×

Text sender sizes are 1.7B / 4B / 8B for Qwen 3 and 3B / 8B / 14B for Ministral 3. Latent senders match the receiver. Speedup is relative to same-size text.

Table 7: Full FanOutQA results for hybrid architectures. Loose and strict accuracy (%) over 50 questions and three receiver draws per setting.

	Gemma 4 Global + sliding-window attention 12B receiver					**Nemotron** Mamba + attention 12B receiver				
	Accuracy ↑		**TTEOA**			**Accuracy ↑**		**TTEOA**		
Setting	Loose	Strict	p50 (s)	p95 (s)	Speedup	Loose	Strict	p50 (s)	p95 (s)	Speedup
Question only	3	1	27	70	–	60	25	8	13	–
Text channels										
Small	75	41	**123**	272	**4.9**×	59	22	**63**	**110**	**2.1**×
Medium	80	53	138	**228**	4.4×	66	24	105	331	1.3×
Same-size	83	52	603	1021	1.0×	73	39	131	251	1.0×
Latent channels										
Full rows	**88**	**62**	363	775	1.7×	Exceeds context limit				
2×	86	61	265	484	2.3×	75	39	140	238	0.9×
4×	**88**	59	201	412	3.0×	**79**	43	109	172	1.2×
8×	85	57	187	364	3.2×	**79**	**50**	102	166	1.3×
16×	87	58	193	353	3.1×	77	45	75	139	1.7×
32×	83	45	200	321	3.0×	**79**	44	80	133	1.6×
64×	71	31	180	302	3.4×	73	31	72	133	1.8×
128×	56	15	178	322	3.4×	57	17	76	129	1.7×

Text sender sizes are E2B / E4B / 12B for Gemma 4 and 4B / 9B / 12B for Nemotron. Latent senders match the receiver. Speedup is relative to same-size text.

E.3 FanOutQA receiver sampling

CacheBack selection is deterministic once the completed sender state and receiver query q are fixed. We therefore hold the completed sender state and selected positions fixed across the three receiver draws and vary only the receiver seed. Text sender messages are likewise fixed across draws. Strict scoring uses deterministic reference matching, so differences across draws reflect receiver generation. We report each draw and the standard deviation across the three draw-level means.

To estimate uncertainty across questions and receiver generations, we use a hierarchical bootstrap with 10,000 samples. Each sample resamples 50 questions and then three receiver draws within each sampled question. We report the 2.5th and 97.5th percentiles as the 95% confidence interval. For CacheBack–text differences, we use the same sampled questions for both settings and resample receiver draws separately within each setting.

Table 8: Receiver sampling on FanOutQA. Strict accuracy (%) over 50 questions and three receiver draws per setting.

Model	Text draws	CacheBack draws	Gain (pp)	95% CI (pp)
Qwen 3 8B	42 / 40 / 40	56 / 52 / 58	+14.7	[1.3, 28.0]
Ministral 3 14B	36 / 36 / 44	62 / 56 / 60	+20.7	[8.7, 32.7]
Gemma 4 12B	54 / 50 / 52	58 / 56 / 64	+7.3	[-3.3, 18.0]
Nemotron Nano 2 12B	44 / 34 / 38	48 / 46 / 56	+11.3	[-0.7, 24.0]
Macro-average	44 / 40 / 44	56 / 53 / 60	+13.5	[6.5, 20.3]

CacheBack uses 4×, 32×, 4×, and 8× for Qwen 3, Ministral 3, Gemma 4, and Nemotron. Family intervals resample questions and receiver draws; the macro-average gives each family equal weight. The macro CI resamples paired questions jointly across families after averaging receiver draws.

E.4 LongBench

Figure 12 shows the full accuracy–latency curves for LongBench v2 Easy. Both relative and fixed CacheBack budgets include settings with higher accuracy and lower latency than same-size text for Qwen and Nemotron.

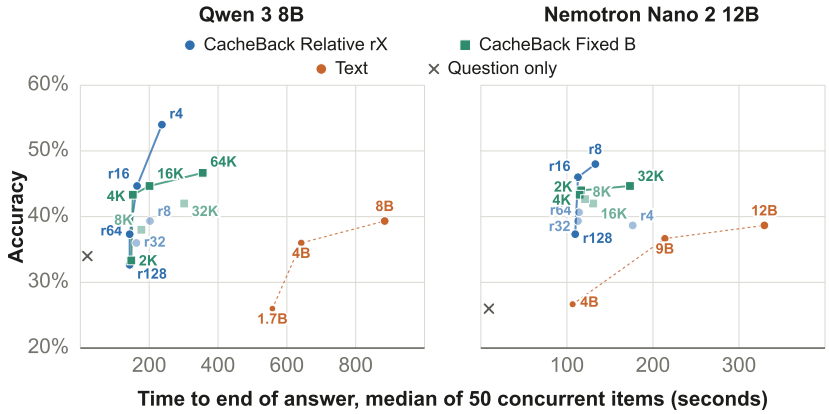

Figure 12: LongBench v2 accuracy and median TTEOA for all evaluated settings. rX denotes relative compression X; xK denotes a fixed budget of x thousand positions.

Table 9: LongBench v2 receiver sampling at selected settings.

Model	Setting	Text draws	CacheBack draws	Gain (pp) [95% CI]
Qwen 3 8B	Relative 4×	36 / 42 / 40	56 / 54 / 52	+14.7 [0.0, 29.3]
	Fixed 64K	36 / 42 / 40	50 / 44 / 46	+7.3 [-8.0, 22.7]
Nemotron Nano 2 12B	Relative 8×	40 / 38 / 38	44 / 48 / 52	+9.3 [-5.3, 23.4]
	Fixed 32K	40 / 38 / 38	38 / 46 / 50	+6.0 [-8.7, 20.7]

Table 9 shows the three receiver draws at selected settings. We use the same sampling and hierarchical-bootstrap procedure as Appendix E.3, holding each message fixed across draws. The selected CacheBack settings improve mean accuracy over same-size text by 6.0–14.7 percentage points.

E.5 LongBench latency breakdown

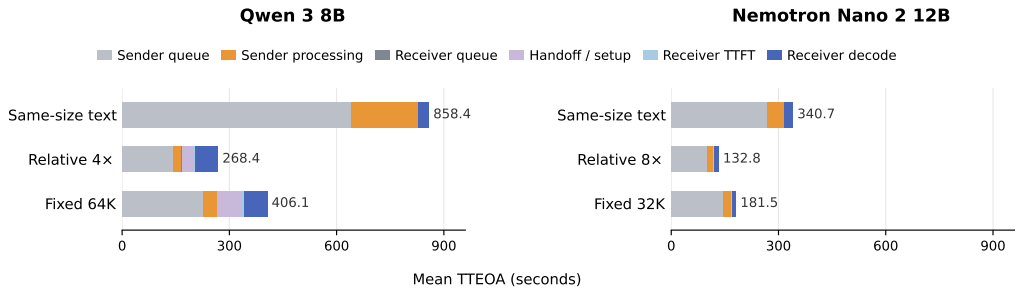

Figure 13: LongBench v2 mean TTEOA by stage at selected settings.

Figure 13 decomposes mean TTEOA by stage. CacheBack mainly reduces sender queueing and processing, with the relative budget faster than the fixed budget.

Table 10: LongBench v2 results on the 50-question Easy panel.

	Qwen 3 8B				Nemotron Nano 2 12B			
	Accuracy ↑	TTEOA			Accuracy ↑	TTEOA		
Setting	(%)	p50 (s)	p95 (s)	Speedup	(%)	p50 (s)	p95 (s)	Speedup
Question only	34	20	74	–	26	9	20	–
Text								
Small	26	558	1014	1.6×	27	107	434	3.1×
Medium	36	642	1137	1.4×	37	214	409	1.5×
Same-size	39	884	1627	1.0×	39	329	693	1.0×
CacheBack, Relative								
4×	54	237	527	3.7×	39	176	288	1.9×
8×	39	202	404	4.4×	48	133	228	2.5×
16×	45	165	276	5.4×	46	113	208	2.9×
32×	36	163	292	5.4×	39	113	199	2.9×
64×	37	143	247	6.2×	41	114	192	2.9×
128×	33	143	245	6.2×	37	110	187	3.0×
CacheBack, Fixed								
64K	47	355	801	2.5×	Exceeds context limit*			
32K	42	302	486	2.9×	45	173	306	1.9×
16K	45	201	414	4.4×	42	130	242	2.5×
8K	38	177	337	5.0×	43	121	218	2.7×
4K	43	152	288	5.8×	43	115	206	2.9×
2K	33	148	274	6.0×	44	117	200	2.8×

Text senders are Qwen 3 1.7B/4B/8B and Nemotron 4B/9B/12B; CacheBack senders match the receiver. Relative uses the listed compression ratio; Fixed caps each message at the listed number of positions. Each sender reads one document quarter.
TTEOA measures submission to the first receiver answer with all 50 questions submitted together. The p50 and p95 are across questions. Speedup is compared to same-size p50.
* For Nemotron, the 64K carried budget plus the next document chunk and prompt exceeds the 131K context window on some questions.

F Serving and evaluation settings

We use the same question panel within each benchmark to compare receiver conditioning across model families and communication topologies. Each run uses one node with eight H100 80GB GPUs and BF16 weights. Models use thinking mode, top-$p = 0.95$, and their recommended sampling settings, with latent vectors rescaled to the mean token-embedding norm. Across latent compression ratios, text sender sizes, two benchmarks, four model families, and three receiver draws, this work used approximately 500 eight-H100 node-hours.

Table 11: Model and serving settings.

	Qwen 3 8B	**Ministral 3** 14B	**Gemma 4** 12B	**Nemotron** Nano 2 12B
Global / total layers	36/36	40/40	8/48	6/62
vLLM	0.11.1	0.26.0	0.26.0	0.26.0
Receiver window	131,072	180,224	262,144	131,072
Temperature	0.6	0.7	1.0	0.6†
Top-k	20	Off	64	Off
Alignment map	Off	Off	Off	On
Position scaling	YaRN 4×	Native	Native	Native

†Nemotron's 4B text sender uses temperature 1.0.

Table 12: GPU allocation by communication setting.

Setting	Sender GPUs	Receiver GPUs
CacheBack, 2×	4	4
CacheBack, 4×–128×	5	3
Text, smaller senders	6	2
Text, same-size senders		8 fused GPUs
Question only	0	8
Full rows, Gemma and Ministral	4	4

We choose the sender–receiver GPU split separately for each communication setting because sender and receiver workloads differ. Using the same split would create different queueing bottlenecks across settings. Each receiver admits up to 16 sequences subject to KV capacity. Prefix caching is disabled. Tables 11 and 12 give the model settings and GPU allocations.

F.1 Timing and output

Time to end of answer (TTEOA) starts when a question is submitted to the node and ends when its first receiver answer finishes. All 50 questions are submitted together, so TTEOA includes queueing for sender and receiver GPUs. We report p50 and p95 over the 50 questions.

We allow receiver generations up to 24,000 tokens, including thinking, to avoid truncating long generations. FanOutQA text senders use the same ceiling. On LongBench v2, text senders use a 16,000-token ceiling because some longer generations repeated the same content; Qwen text senders also use a presence penalty of 1.5. Before each run, we verify that the receiver prompt, incoming messages, and 24,000-token receiver output ceiling fit within the context window. For Qwen 3 and Nemotron, the Full rows FanOutQA control exceeds that window.

F.2 FanOutQA evaluation and scoring

We evaluate 50 questions from the FanOutQA development split, excluding the questions used for selector development in Section 3.2. Each question must provide at least 40,000 tokens of context to every sender under the Qwen 3 tokenizer. We order the remaining questions by the SHA-256 hash of the question ID and take the first 50. We replace one question because source truncation removes a required reference string from the sender context.

We score receiver answers by matching the FanOutQA reference groups. Strict accuracy requires a match for every reference group in the answer after normalization. Loose accuracy is the fraction of matches. We average the three receiver draws for each question, then average across the 50 questions.

Before the final runs, we checked all 498 reference strings against their evidence pages. We corrected nine strings in seven questions: three typos, three values that disagreed with the page, and three answers unsupported by the page. We also match a small set of equivalent forms, including 72 and 72.0, km2 and km^2, and one shared-surname case that still requires both people. Every model and setting uses the same references and matching rules.

We rescore all 6,900 saved answers from the 46 settings using the original FanOutQA references. CacheBack remains more accurate than same-size text at the reported operating points for all four model families (Table 13).

We apply the same matching rules to sender messages. Across the 50 questions, there are 495 audited reference groups. A group counts as present if any of the three sender messages contains an accepted form.

For text, we search only the message sent to the receiver, not the sender's private thinking. For CacheBack, we decode the retained source positions and search the resulting text. A match must come from one sender without crossing discarded positions. We exclude generated latent positions and include prompt framing.

Appendix D.1 instead measures evidence recall on the selector-development questions.

Table 13: FanOutQA strict accuracy (%) before and after the reference audit.

		Text			CacheBack		
Family	Ratio	Before	After	Change	Before	After	Change
Qwen	4×	30.0	40.7	+10.7	41.3	55.3	+14.0
Ministral	32×	32.0	38.7	+6.7	46.7	59.3	+12.7
Gemma	4×	36.7	52.0	+15.3	46.0	59.3	+13.3
Nemotron	8×	32.0	38.7	+6.7	38.7	50.0	+11.3

F.3 LongBench evaluation and scoring

We evaluate 50 LongBench v2 questions with documents between 100K and 250K tokens. We use at most one question per document and select the panel by document length and task family using a seeded hash. In an initial mixed-difficulty pass, the same-size Qwen 3 8B text sender scored 36.2% on Easy questions and 25.9% on Hard questions, while 46% of its reports reached the length ceiling. These reports can end before the sender finishes its notes, making the comparison depend on the report ceiling. For simplicity, we restrict the final panel to Easy questions, add the instruction "Keep the notes under 8,000 words." to the text prompt, and, for Qwen only, use a presence penalty of 1.5. Only 8% of Qwen 3 8B text reports then reach the ceiling.

Each LongBench v2 document is split into quarters. The first agent reads the first quarter. Each later agent receives the previous message, reads the next quarter, and sends a new message. The receiver answers the multiple-choice question from the fourth message.

For text, the fourth message is a text message. For CacheBack, it contains the fourth sender's selected positions. The question-only setting answers from the question and choices alone. Appendix G.2 gives the exact prompts.

We score the text after the final `</think>` tag. We remove asterisks and extract A, B, C, or D from either `The correct answer is (X)` or `The correct answer is X`. The draw scores one if the extracted letter matches the reference answer and zero otherwise. If neither form appears, it also scores zero. We average the three receiver draws for each question and then average across questions. cross 4,650 receiver generations, 99 required an inserted `</think>` tag followed by continuation; 76 of these had reached the initial output limit. We allow one continuation of at most 4,096 tokens under the same seed, which lets generation continue into the visible answer.

G PROMPTS AND INPUT CONSTRUCTION

G.1 FANOUTQA PROMPTS

All FanOutQA settings use the same sender evidence and question. Text messages fill the receiver's report slots; CacheBack supplies continuous rows at the positions described in Appendix G.3. The receiver question and answer instruction are unchanged.

Sender

```
You are one reasoning worker.  Inspect only your private
evidence.  Work through every relevant name, number,
relationship, and uncertainty that could help the coordinator
answer the question.  Return a complete reasoning handoff,
including intermediate reasoning and unresolved alternatives.
Do not claim access to any other worker's evidence.

Private evidence:
{private evidence}

Question:  {question}

Return the complete reasoning handoff now.
```

Receiver

```
You are the coordinator.  Answer from the supplied handoff and
do not invent missing evidence.

Worker reports:
[worker 0 report]
{report 0}

[worker 1 report]
{report 1}

[worker 2 report]
{report 2}

Question:  {question}

Give the answer directly and include every requested leaf.
```

Ministral replaces only the receiver's opening sentence with the following.

```
You are the coordinator.  Answer from the supplied handoff;
invent no evidence.
```

G.2 LONGBENCH PROMPTS

Text and CacheBack use the same question, choices, and final receiver prompt. Text senders pass messages through the chain, while CacheBack passes selected source and latent positions.

Text sender

```
You are one reading worker in a chain of 4.  You receive the
running notes from the earlier parts of a text and one new
part.  Rewrite the notes into a compact summary of everything
read so far that bears on the question below:  the names,
numbers, dates, events, relationships, and open alternatives it
turns on, each stated once in your own words.  Do not copy
passages, code, tables, or data from the part; state what they
say and where they sit.  Drop what the question does not need.
Keep the notes under 8,000 words.  Return only the rewritten
notes, not a final answer.

Question and choices:
{question}
Choices:
(A) {choice A}
(B) {choice B}
(C) {choice C}
(D) {choice D}

Notes from the earlier parts:
{running notes}

Part {hop} of 4:
{current text chunk}

Return the rewritten notes now.
```

Final receiver

```
The notes a chain of readers kept from the document are below.
Answer from them, not from what you already know.

<text>
{final notes}
</text>

What is the correct answer to this question:  {question}
Choices:
(A) {choice A}
(B) {choice B}
(C) {choice C}
(D) {choice D}

Format your response as follows:  "The correct answer is
(insert answer here)".
```

G.3 CHAT-TEMPLATE PACKING

The four model families use different native prompt formats. We preserve each model's format and document where CacheBack rows are inserted. The same packing is used across settings within each benchmark and model family.

For Qwen, CacheBack rows precede the complete rendered prompt. On LongBench, this applies to readers 2–4 and the final receiver; reader 1 has no incoming message.

For Gemma, which we evaluate only on FanOutQA, CacheBack rows precede the complete rendered receiver prompt, including Gemma's native turn framing.

For Nemotron, FanOutQA inserts CacheBack rows into the report slots inside the receiver's user turn. On LongBench, incoming rows precede the complete rendered prompt for readers 2–4. For the final receiver, the rows are inserted inside the user turn after the opening sentence and payload header.

For Ministral, which we also evaluate only on FanOutQA, CacheBack rows precede the complete rendered receiver prompt. Receivers append the following instruction below to the supplied system prompt, while senders use the original system prompt.

```
It is imperative to close the [THINK] tag with a [/THINK] once
you are ready to present the answer to the user.
```