CONSTRUCTOR

Link: Teaching Attention to Have Intention on Long-Context Tasks

Constructor Team

research@constructoragi.com · Technical Report

Abstract

Agents that run for weeks accumulate histories of millions of tokens, and the detail they need next may be a single line written near the start. Full attention can recover such a line, but its cost grows quadratically with the history, and fixed-size recurrent states cannot retain it verbatim. We present Link, a learned sparse read that replaces the full read in the attention layers of Qwen3.5-9B and leaves its linear layers unchanged. For each query, a small router selects a fixed number of chunks from a key-value store that can reside outside GPU memory, so each query reads the same amount however long the history grows. We compare Link with the model's own attention, extended with YaRN as its model card prescribes, after identical task training on 1M-token coding-agent logs. On held-out tasks, Link answers 69.5% of verbatim-recall questions at 1M tokens versus 56.6%, and 55.5% versus 36.3% at 4M tokens, while reading only 0.16% of the input. It processes a 4M-token log in under six minutes on one H100, using at least 9.9× less GPU time than full attention.

1 Introduction

Consider an agent that has spent weeks on one codebase, running thousands of commands and watching the tests fail in ways nobody wrote down. When a new failure appears, the most useful part of that history can be one line, millions of tokens back: the exception the same test raised a week earlier, or the file the agent opened first the last time it touched the module. A summary of the history may well have dropped that line, so the agent has to recover it verbatim, and cheaply enough to run on one GPU.

Current architectures make this expensive or lossy. Full softmax attention (Vaswani et al., 2017) can reach any earlier token, but its compute grows quadratically with length and its key-value cache linearly. The refinements now standard in open models ease these costs without removing them. Grouped-query attention (GQA; Ainslie et al., 2023) shares keys and values across query heads and shrinks the cache by a constant factor, and YaRN (Peng et al., 2024) rescales rotary positions (Su et al., 2021) so that attention reaches past the trained window, with a scale factor of 16 at 4M tokens for a native window of 262,144. Even with both, in our experiments the full-attention read of Qwen3.5-9B fits on one H100 at 1M tokens but not at 2M or 4M. Linear-attention and state-space layers (Katharopoulos et al., 2020; Gu and Dao, 2024; Yang et al., 2025) run at constant cost per token, but a fixed-size state cannot hold the verbatim content of an arbitrarily long history. Hybrids such as Qwen3.5 (Qwen Team, 2026) interleave the two kinds of layer, and their attention layers still pay the full cost of long inputs.

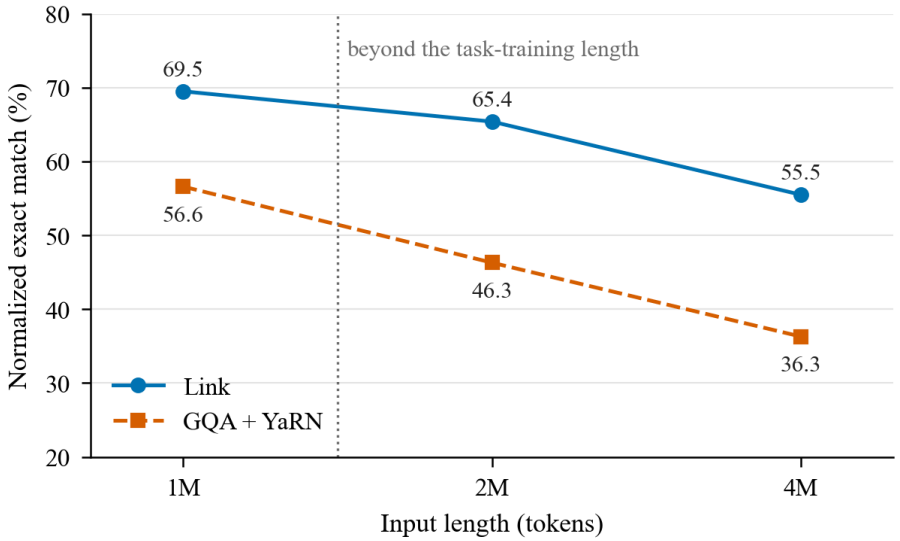


Figure 1: Normalized exact match on held-out test sessions by input length, three-seed means per system after identical task training; 32 sessions and 5,039 questions at 1M, 16 and 2,048 at 2M, 8 and 1,024 at 4M. The dotted line separates the 1M-token length of both systems' task training from the longer inputs. Figure 3 gives the 95% intervals on Link's lead.

Link is a targeted change to an existing model: it alters only the quadratic attention layers of Qwen3.5-9B, giving them a way to choose what they read. Each such layer attends in full to a short prefix and a recent window, keeps the rest of the context as fixed-size chunks of key-value cache, and lets a small learned router pick a sparse set of those chunks for each query. Once the chunk store outgrows GPU memory it lives in host memory. Selection happens inside attention, over key-value states the model has computed from the whole input, so the mechanism sits between dense attention and retrieval outside the model.

We evaluate on the setting that motivates this work, exact recall from long agent logs. The baseline is not vanilla attention but the model's own attention as it is meant to run on long inputs: grouped-query attention extended with YaRN, following the Qwen3.5 model card, with the same Gated DeltaNet layers as Link. We refer to it as GQA + YaRN. Both systems receive identical task training on 1M-token sessions, and we test both on tasks and repositories held out from task training. Every answer is a file path or a whole line copied verbatim from the log, and it never lies inside the span Link reads in full. Figure 1 shows the result. Link scores 69.5% normalized exact match at 1M tokens against 56.6% for the baseline, a lead of +12.9 [+2.6, +23.1] points (95% interval in brackets; leads are computed before rounding), with Link ahead in all 32 sessions and for all nine pairings of training runs. The interval is conservative: with three training runs per system its multiplier is 2.8 to 4.3, against 1.96 for a normal interval (Section 5.5). The lead is +19.1 [+6.1, +32.1] at 2M and +19.2 [+9.0, +29.3] at 4M, beyond the length either system was trained on.

At 4M the median answer lies about four times farther back than at 1M. From 1M to 4M, the dense baseline falls from 56.6 to 36.3 and Link from 69.5 to 55.5. Figure 2 shows the cost of each read. At 4M, Link's median read is 9.9× faster than that of full attention, and the gap grows with length: 5.7× at 2M and 3.9× at 1M, where both systems use one GPU and Link's store is still on the GPU, so no offloading is involved.

The lead holds on test tasks that appear nowhere in the data used to train or validate Link. On these never-seen tasks Link leads by +12.4 points at 1M, +17.1 at 2M and +18.1 at 4M, and all six intervals in Figure 3 exclude zero.

In summary, this paper shows the following.

- **A read that does not grow.** In our agent-log experiments each query head reads about 6,200

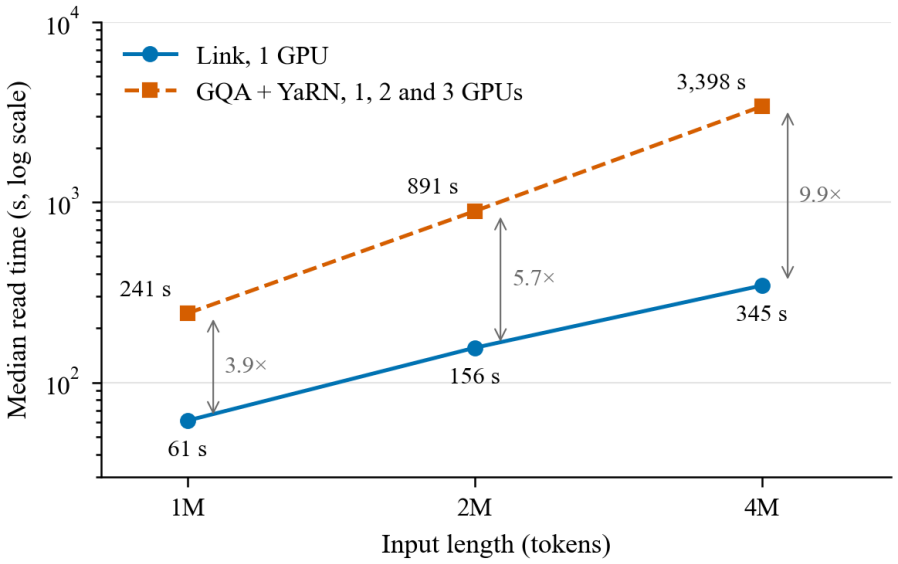


Figure 2: Median time to read one test session, from the seed-0 models of both systems at the end of task training, over 32, 16 and 8 sessions. Link reads every session on one H100. GQA + YaRN ran on one, two and three GPUs at 1M, 2M and 4M tokens, with its layers split across them and run one after another. Speedups are ratios of the unrounded medians.

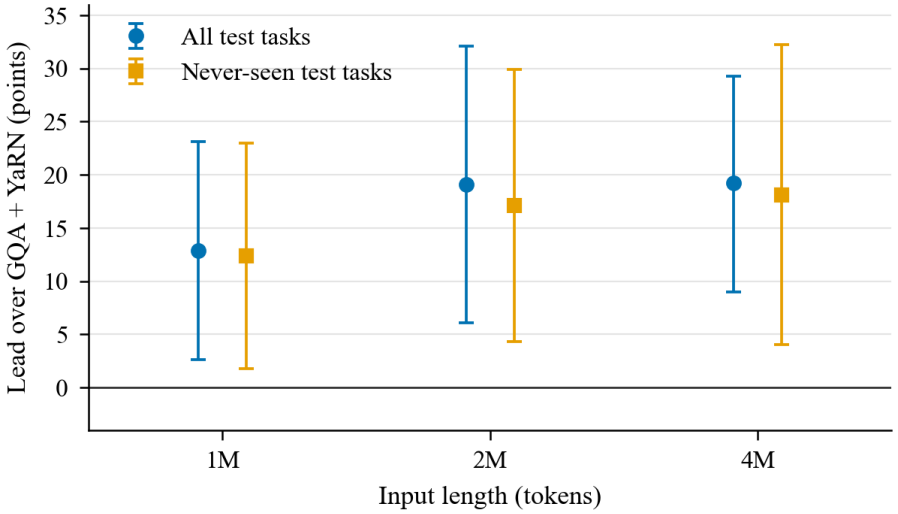


Figure 3: Link's lead over GQA + YaRN, in points, with 95% intervals that account for both training runs and test sessions, on all test tasks and on the never-seen subset (test tasks that appear nowhere in the data used to train or validate Link). Questions per point (all, never-seen): 5,039 and 2,285 at 1M, 2,048 and 896 at 2M, 1,024 and 478 at 4M.

Table 1: Results at a glance on the held-out coding-agent logs. Scores are normalized exact match (%), three-seed means after identical task training; reinforcement learning is further training of Link alone (Section 6.6). Read times are medians over the test sessions for the seed-0 models (Section 7).

	1M	2M	4M
Share of the input one Link query head reads	0.6%	0.3%	0.16%
Link	**69.5**	**65.4**	**55.5**
GQA + YaRN, which reads every position	56.6	46.3	36.3
Keyword retrieval (BM25)	49.5	44.0	37.3
Link's lead over GQA + YaRN (points)	+12.9	+19.1	+19.2
Sessions led by Link	32 of 32	16 of 16	8 of 8
Link after reinforcement learning	77.8	72.9	66.3
Read time, Link	61 s, 1 GPU	156 s, 1 GPU	345 s, 1 GPU
Read time, GQA + YaRN	241 s, 1 GPU	891 s, 2 GPUs	3,398 s, 3 GPUs
Wall-clock speedup	3.9×	5.7×	9.9×

positions at any length: 0.6% of a 1M-token input and 0.16% of a 4M one. A small router, trained to predict where the model's own attention places its mass, chooses them, and the store can leave the GPU.

- **More accurate than its own full attention.** Under identical training, Link leads the model's own full attention by 12.9, 19.1 and 19.2 points at 1M, 2M and 4M tokens; at 4M that is 55.5% against 36.3%, or 53% more questions answered. It leads in every test session at each length, its weakest training run is ahead of the baseline's strongest, and it leads identically trained BM25 retrieval by about 20 points at every length.
- **Four times past its training length.** Both systems train at 1M tokens, and at 4M the baseline runs with its YaRN factor raised to four times its trained setting. At 4M, Link scores about what full attention scores at 1M (55.5% against 56.6%), and its router finds the answer's chunk as often 3.7M tokens back as nearby.
- **A fraction of the cost.** At 4M, Link reads a session in under six minutes on one GPU, while full attention takes 57 minutes on three GPUs: at least 9.9× less GPU time (Section 7). Link's peak GPU memory stays near 24 GiB from 2M to 4M.
- **Controls against flattering a sparse read.** We score only answers that lie outside the span Link reads in full. Leakage filters, a memorization control (0.2% correct without the log), a never-seen subset, three seeds per system and session-level statistics guard the comparison, and Table D.3 records when each evaluation choice was fixed.
- **Headroom.** With the same read, further training of Link alone raises it further: reinforcement learning on its selections and answers reaches 66.3% at 4M, and a single longer training run 77.0%.
- **Inside complete systems.** On the public BABILong benchmark at 512K and 1M tokens, a system built on Link leads by 8.0 and 6.2 points over the same model trained on the same questions for the same number of steps and answering from keyword-retrieved text, and a LongBench v2 system built on the 9B model scores 61.2% (Section 8).

Table 1 collects the main results at the three lengths.

Sections 3 and 4 describe the read and its training, Section 5 the evaluation protocol, Sections 6 and 7 the results and their cost, and Section 8 two systems built on Link for public benchmarks. The appendix holds the procedures, derivations and protocol details behind each of these sections.

2 Related Work

Work on long inputs has followed two broad routes. The first keeps attention dense and improves it: grouped-query attention (Ainslie et al., 2023) shares keys and values across query heads to shrink the cache, and positional interpolation and YaRN (Chen et al., 2023; Peng et al., 2024) let a model attend past its trained window. Compute and memory still grow with the input. Our baseline combines both. The second route limits what each query reads. Memory-based models attend to stored past states (Dai et al., 2019; Wu et al., 2022; Bertsch et al., 2023; Mohtashami and Jaggi, 2023). Training-free methods such as StreamingLLM (Xiao et al., 2024b), H2O (Zhang et al., 2023), SnapKV (Li et al., 2024), InfLLM (Xiao et al., 2024a), Quest (Tang et al., 2024) and MInference (Jiang et al., 2024) evict from the key-value cache or attend to a sparse part of it at inference time, and several systems, among them InfLLM, ShadowKV (Sun et al., 2025), RetrievalAttention (Liu et al., 2025) and InfiniteHiP (Lee et al., 2025), offload part or all of that cache to host memory. More recent work builds the selection into the model and trains it, as in MoBA (Lu et al., 2025), SeerAttention (Gao et al., 2024), Native Sparse Attention (Yuan et al., 2025) and DeepSeek Sparse Attention (DeepSeek-AI, 2025). Retrieval-augmented generation (Lewis et al., 2020) sits outside the model altogether: it selects passages, often by lexical scoring such as BM25 (Robertson and Zaragoza, 2009), and passes them in as text.

Link follows the second route and differs from each of these methods in choices aimed at multi-million-token inputs. Its structure, an exact prefix and window around a selectable middle, resembles InfLLM's, yet the selection is learned rather than fixed. Its chunk summaries are built from key statistics, as Quest's are, but they are trained against the model's own attention instead of serving as a hand-designed bound. MoBA and DeepSeek Sparse Attention also learn which blocks to read; Link differs in scoring blocks only on position-free coordinates, and, unlike the trained methods above as published, it keeps its store in host memory. Because it selects from key-value states computed over the whole input, it also has access to context that a retriever working on raw text cannot see. Together, these choices give a read whose attention cost and GPU memory stay bounded at any input length, and whose selection, as Section 6 shows, keeps working beyond the lengths it was trained on. Our comparisons are with the two systems Link lies between, the model's own attention and retrieval from the raw text, each given identical task training; we have not run the sparse-attention methods above under the same training.

3 Model Architecture

Qwen3.5-9B interleaves softmax-attention layers with Gated DeltaNet layers (Yang et al., 2025), which summarize the input in a fixed-size recurrent state. Link confines its change to the quadratic layers. In each of the model's eight softmax-attention layers, it replaces the read of every earlier position with a bounded selection; the linear-time Gated DeltaNet layers are left as they are and still process the entire input.

3.1 Full Attention and Its Cost

Consider an attention layer with H query heads and H_k key-value heads of dimension d. Under grouped-query attention, each group of H/H_k query heads shares one set of keys and values. Appendix A defines the terms and symbols used from here on. For a query q_t at position t, one head of full attention computes

$$a_t = \sum_{i \le t} \alpha_{t,i}\, v_i, \qquad \alpha_{t,i} = \frac{\exp\left(q_t^\top k_i / \sqrt{d}\right)}{\sum_{i' \le t} \exp\left(q_t^\top k_{i'} / \sqrt{d}\right)}. \tag{1}$$

Reading a session of n tokens therefore requires

$$C_{\text{full}}(n) = \sum_{t=1}^{n} t = \frac{n(n+1)}{2} = \Theta(n^2) \tag{2}$$

key-query inner products per query head, and each layer holds a key-value cache of

$$M_{\text{full}}(n) = 2\, n\, H_k\, d = \Theta(n) \tag{3}$$

entries. Grouped-query attention reduces (3) by the constant factor H/H_k and leaves the growth rate unchanged. The linear growth of the cache is what forces full attention onto more GPUs: its read of Qwen3.5-9B fits on one H100 at 1M tokens but not at 2M or 4M (Section 7). Reaching past the native window $N_0 = 262{,}144$ also requires rescaling the rotary positions (Su et al., 2021). YaRN (Peng et al., 2024) does so with the factor

$$s = N/N_0 \tag{4}$$

for a target length N, which gives $s = 4$, 8 and 16 at 1M, 2M and 4M tokens. Every query still reads every earlier position, now through a stretched position encoding.

3.2 The Read

Link splits the positions up to t into three regions: a short prefix P at the start, a recent window W_t of the latest positions ending at t, and between them consecutive chunks of equal length L, $C_1, C_2, \ldots$, whose keys and values form the chunk store. The prefix and the window form the exact window, which is always read in full. For a set S_t of k selected chunks, the query reads

$$R_t = P \;\cup\; \bigcup_{j \in S_t} C_j \;\cup\; W_t, \tag{5}$$

and attention runs the usual softmax of (1) with the sums restricted to R_t. Positions outside R_t receive $\alpha_{t,i} = 0$. Since k is fixed, the read is bounded at any input length:

$$|R_t| \le |P| + kL + |W_t| =: B. \tag{6}$$

In the coding-log experiments of Sections 6 and 7, B is about 6,200 positions per query and key-value head, the same in every run and at every length: about 0.6% of a 1M-token input, 0.3% at 2M and 0.16% at 4M. A read that does not grow with the input is what bounds the attention cost of each query and the GPU memory it needs (Sections 3.4 and 3.5); Link's lead at longer inputs therefore comes from where it reads, not from reading more. Figure 4 summarizes the read, and Algorithms B.1 and B.2 give it step by step.

3.3 Selection

A router with its own parameters ϕ scores every stored chunk for each query and key-value head, and each head keeps its k highest-scoring chunks. Qwen3.5 applies rotary encoding to only part of each head's coordinates. We write $\tilde{k}$ and $\tilde{q}$ for keys and queries restricted to the remaining, position-free coordinates, and $\tilde{K}_{h,j}$ for the position-free keys of chunk j in key-value head h. The router summarizes each chunk with pooled statistics of its keys,

$$s_{h,j} = W_h^K\, \psi\big(\tilde{K}_{h,j}\big) + b_h^K, \qquad \psi(\tilde{K}) = \Big[\tfrac{1}{L}\textstyle\sum_i \tilde{k}_i\,;\; \max_i \tilde{k}_i\,;\; \min_i \tilde{k}_i\Big], \tag{7}$$

with maxima and minima taken coordinatewise, and maps the $G = H/H_k$ queries that share the head into the same r-dimensional space:

$$u_{h,t} = \frac{1}{\sqrt{r}}\Big(W_h^Q\,\big[\tilde{q}_{t,h,1};\,\ldots;\,\tilde{q}_{t,h,G}\big] + b_h^Q\Big). \tag{8}$$

The score and the selection are

$$\sigma_{t,h}(j) = u_{h,t}^{\top} s_{h,j}, \qquad S_{t,h} = \operatorname*{top\text{-}k}_{j}\; \sigma_{t,h}(j), \tag{9}$$

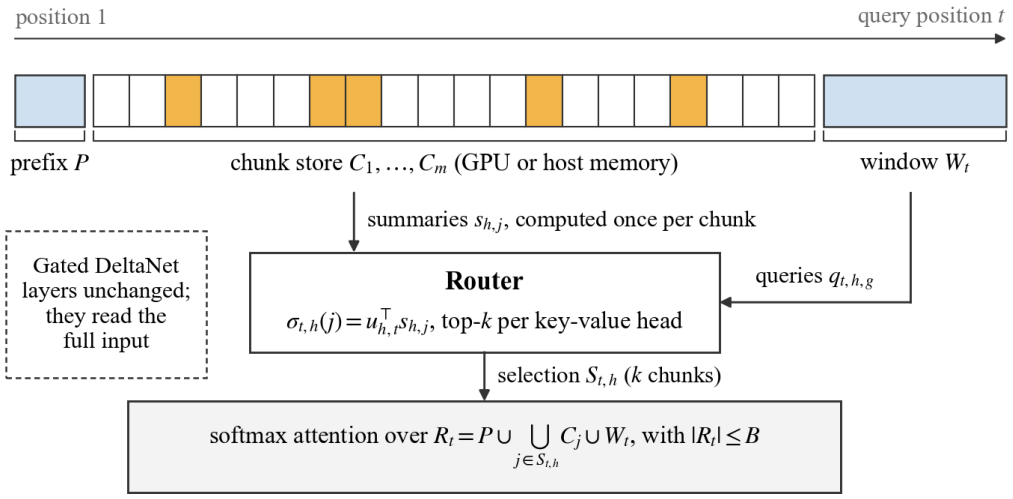


Figure 4: The Link read in one attention layer. The exact window (blue) is always read in full. The rest of the input is kept as fixed-size chunks of key-value cache, on the GPU or in host memory, and a small learned router selects a sparse set of them for each query (orange), the same number at any input length. The Gated DeltaNet layers are unchanged and read the full input.

and the read (5) of every query head in the group uses $S_{t,h}$.

The design follows from one constraint: the read has to keep working far beyond the lengths it was trained on. Task training uses 1M-token sessions, and we test at 4M. A scorer that used the position-encoded coordinates would meet rotary phases at 4M that it never saw in training, and YaRN rescaling would shift them again. Scoring only the position-free coordinates removes that dependence, so a chunk three million tokens back is judged on what it contains, exactly as it would be at a distance of a thousand tokens (Appendix B.1). The design relies on this split, which Qwen3.5 shares with other models that rotate only part of each head; a model that rotates every coordinate would need another position-free signal. Position still matters inside the read, where the model's own attention runs over the selected chunks. Section 6.5 measures the behavior this choice is meant to produce: at 4M tokens the router selects the answer's chunk as often 3.7M tokens back as nearby, almost four times farther than any distance in task training.

The other choices concern cost. Scoring happens per key-value head because that is the unit in which the cache is stored: under grouped-query attention one selection serves all G query heads of a group, and with the store in host memory it also fixes how much data crosses to the GPU. A chunk's summary depends only on its own keys, so it is computed once, when the chunk enters the store, and reused by every later query. Scoring a chunk then costs one r-dimensional inner product instead of attention over its L keys. The coordinatewise extrema sit beside the mean so that a chunk holding one strongly matching key is not averaged away by its neighbors, as it would be under a centroid alone; Appendix B.2 makes the argument precise.

We initialize the router so that $\sigma_{t,h}(j)$ equals the inner product between the group's queries and the chunk's mean key under a random orthonormal projection, which estimates the attention logit to the chunk's centroid (construction in Appendix B.3). Training (Section 4.1) turns that estimate into a learned predictor of where full attention places its mass.

At inference the selection is deterministic but depends on the query, so two questions about the same log can draw on different regions of the store. This is also where selection parts ways with retrieval. A retriever ranks raw text outside the model, whereas the router ranks keys that the model has itself computed over the whole input, in the representation attention uses, and it does so afresh at every query, in every attention layer.

The selection in (9) is discrete. A chunk outside S_t contributes nothing to a_t, so for $j \notin S_t$ and $i \in C_j$,

$$\frac{\partial a_t}{\partial v_i} = 0, \tag{10}$$

and the language-model loss carries no gradient for chunks the read leaves out. This property shapes how the read can be trained (Sections 4 and 6.6).

3.4 Cost

From (6), attention over a session costs

$$C_{\text{attn}}(n) \le nB = \Theta(n) \tag{11}$$

inner products per query head, against $\Theta(n^2)$ in (2). Attention for each query therefore costs the same at 4M tokens as at 1M, and the share of the history it reads, B/n, halves each time the input doubles. The router's scoring remains linear, per query, in the number of stored chunks $m = \lfloor (n - |P| - |W_t|)/L \rfloor$, so its total work is $\Theta(nm)$. Because it scores a compact summary of each chunk and has few parameters next to the 9B base model, this term stays small over the lengths we test: measured read time grows close to linearly from 2M to 4M tokens (Section 7). Appendix B.4 shows that the router's quadratic term is a fixed fraction, $r/(2GLd)$, of the work of full attention at any length.

3.5 The Store

Once the store outgrows GPU memory, Link keeps it in host memory and moves only selected chunks to the GPU. By (6), the key-value state attention needs on the GPU is then bounded independently of n, and only the host store grows with the input:

$$M^{\text{kv}}_{\text{GPU}}(n) = O(1), \qquad M^{\text{kv}}_{\text{host}}(n) = \Theta(n). \tag{12}$$

Section 7 confirms this behavior: peak GPU memory barely moves when the input doubles from 2M to 4M tokens. Table B.2 itemizes what each system keeps, and where.

4 Training

4.1 The Router

We train the router first, in a run of its own, by self-distillation from the attention of the unmodified model, in the manner of the indexer warm-up of DeepSeek Sparse Attention (DeepSeek-AI, 2025) and the attention gate of SeerAttention (Gao et al., 2024). The target for query position t and key-value head h is the share of the model's causal attention that falls in each stored chunk, summed over the G query heads of the group:

$$p_{t,h}(j) = \frac{\sum_{g=1}^{G} \sum_{i \in C_j} \alpha^{(g)}_{t,i}}{\sum_{j'} \sum_{g=1}^{G} \sum_{i \in C_{j'}} \alpha^{(g)}_{t,i}}, \tag{13}$$

where $\alpha^{(g)}_{t,i}$ are the attention weights (1) of query head g, computed over all of its coordinates, position-encoded ones included. The router minimizes

$$\mathcal{L}(\phi) = \mathbb{E}_{t,h}\Big[\, w_{t,h}\ \mathrm{KL}\big(p_{t,h} \,\big\|\, \mathrm{softmax}_j\, \sigma_{t,h}(j)\big)\Big] \tag{14}$$

with $w_{t,h}$ proportional to the attention mass the query places on the store, so that queries which attend mainly to their local context weigh less (Appendix C.1 defines the weights). The router therefore learns to predict from position-free coordinates where full attention would place its mass. Every Link model in the main comparison starts from this one router. That run used none of our questions, and no answer label entered it or the choice of its checkpoint.

4.2 Task Training

Link and the baseline start from Qwen3.5-9B and receive identical task training: the same trainer code, training sessions and data, and optimization settings. A Link run and a baseline run with the same seed train on the same sessions in the same order, step for step, and their recorded run configurations differ in two things: the read itself and how many GPUs training uses (Figure C.1). Each run keeps the checkpoint with the highest exact match on one fixed sample of separate validation sessions, the same sample for every run of both systems, with ties going to the earlier one. This rule is recorded in each run's configuration, and the chosen weights are frozen before testing.

The baseline leaves the model's attention unmodified: every query attends over the whole input through the model's built-in grouped-query attention, with positions extended by YaRN. Its scale factor follows (4): 4 at 1M, the setting the Qwen3.5 model card gives for extending the window to about 1M tokens (Qwen Team, 2026), then 8 at 2M and 16 at 4M, so every test session lies inside the stretched window. The factors were fixed by this rule before full attention had been run at 2M or 4M (Table D.3). The baseline was task-trained with factor 4, so at 1M it runs exactly as trained, and Link already leads there by +12.9 points (Section 6.1); at 2M and 4M it runs with factors its task training never used. Link applies no YaRN rescaling at any length.

We train each system with three seeds. Seed 0 is the original pair of runs, and seeds 1 and 2 are replications. Before training each of them, we wrote down the rule for the lead to stand at a length: every pairing of a Link seed with a baseline seed must favor Link, and the interval of the seed-averaged lead must exclude zero. All task training uses 1M-token sessions, so at 2M and 4M each system runs beyond its task-training length, with the same weights it uses at 1M.

4.3 Further Training

After task training we extend Link's training in two ways, reinforcement learning and additional task training, and we report those results separately (Section 6.6). The main comparison, by contrast, isolates the read. Link and the baseline share their base model, task training, tokens and decoding; what separates them is the read itself, together with the router Link trains beforehand.

5 Evaluation

Earlier evaluations found that the effective context of many models falls well short of their stated window (RULER; Hsieh et al., 2024) and that models make poor use of the middle of long inputs (Liu et al., 2024b). Benchmarks such as BABILong and LongBench v2, to which we return in Section 8, test reasoning over long inputs. Our own benchmark asks a narrower question, whether a model can return verbatim text from agent logs of 1M to 4M tokens, and scores the answer by normalized exact match. Its questions have one checkable answer and measure in-context recall: whether the model can recover a line from its own context, not whether a search tool, which an agent can also call, would find it. Task training sees no test task and no repository of a scored question, and without the log a trained model answers 0.2% of the questions (Section 5.4), so a correct answer comes from the log being read. The router's own training run used no questions or answers but read logs that include about half of the test tasks; the never-seen subset (Section 6.3) leaves those tasks out, and Link's lead holds on it.

Sparse reads are easy to evaluate in ways that flatter them, and the protocol guards against four such pitfalls. An answer near the end of the input falls inside the recent window, which Link reads in full, so we call a question eligible when every occurrence of its answer lies outside Link's exact window, and we score no other. Questions about public text can be answered from memory, so we drop test questions that leak from the training data and run a memorization control. A baseline trained on other data or decoded another way would mix those differences into the comparison, so both systems get the same task training on the same sessions, the same tokens and the same decoding. Finally, questions drawn from one log are correlated, which makes the session our sampling unit. We wrote down the definition of eligible questions, the scoring rule, the leakage filters and the task-training checkpoint

Table 2: What could flatter a sparse read, and the control against it.

What could flatter a sparse read	Control
The answer sits in the span Link reads in full	Only questions whose every answer occurrence lies outside that span are scored (Section 5.3)
The model already knows the answer	Leakage filters; without the log, a trained model answers 0.2% of the questions (Section 5.4)
The baseline is trained or decoded differently	Same trainer, sessions, session order, tokens and decoding; three seeds per system (Section 4.2)
One favorable training run	All 27 pairings of a Link run with a baseline run favor Link (Section 6.1)
Correlated questions overstate confidence	The session is the sampling unit, with an exact sign test over sessions (Section 5.5)
Test tasks that appear in the data used to train or validate Link, the router's run included	The never-seen subset, on which the lead holds (Section 6.3)
Choices made after seeing results	Table D.3 records when each evaluation choice was fixed

rule before task training of the compared models began; Table D.3 records when each of the other evaluation choices was fixed. The size of Link's read was settled in exploratory development runs, some of them scored on the 1M test sessions, and is the same in every compared run and at every length. Table 2 sets each pitfall beside its control, with three further guards.

5.1 Sessions and Questions

Each input is the log of a software-engineering agent working through a long sequence of issues from real repositories. A session concatenates many recorded agent episodes, one per issue, and neighboring tasks often look alike. For each task the log records the issue text, the agent's messages with the commands it ran, and the commands' outputs.

After an end-of-log marker comes one question about one earlier task. It names the task by its exact title, by the first file the agent opened in it, or as the task immediately after a titled one. Naming a task by its file makes the model find it by a detail inside its body, and naming it as the next task adds a second hop. The question then asks for one of three things from that task: the path the agent gave when it first opened a file, the first exception line printed, or the first non-blank line of its first real code edit. An exception line reads `XxxError: message` or `XxxException: message`. We ask for the edit line only when it meets a fixed minimum length and is not a comment or docstring line, and a question that names its task by the first file opened in it never asks for that file's path.

Titles are unique within a session, and the file that names a task occurs nowhere else in it, so each question identifies exactly one task. No question's text contains its answer, or any shorter form of a path answer that scoring accepts. Because all episodes share one agent scaffold, the log is full of lines shaped like the answer, while each gold answer occurs inside its own task alone. Table D.1 lists the seven question types that occur, Figure D.2 shows the layout of a prompt, and Figure F.1 gives both systems' scores on each type.

5.2 Scoring

Both systems see the same plain-text prompt, with no chat template: the log's one-line header, the log, then a question block that ends with the line "Reply with only the answer, on a single line." and a line "Answer:". Both decode greedily with Qwen3.5's thinking mode off and stop at the end of the first answer line, under a shared token budget that is always longer than the gold answer. All questions about a log are answered on one read of it, which is restored after every answer, so no question sees another question or its answer.

The gold answer is copied verbatim from the log: a file path, or a whole line for the exception and edit questions. We take the reply's first line, strip surrounding whitespace, quotes, backticks and trailing

Table 3: Test sets. Answer distance, in tokens, runs from the answer's nearest occurrence in the log to the question, which keeps distances conservative when an answer appears more than once within its task.

Length	Sessions	Tokens per session	Tasks per session	Median answer distance
1M	32	0.94M–1.00M	55–79	508K
2M	16	1.89M–1.98M	116–140	1.03M
4M	8	3.84M–3.95M	246–269	1.98M

periods, collapse internal runs of whitespace to one space, and require an exact, case-sensitive match. No model acts as a judge. A path also counts in a longer or shorter form that keeps at least its last directory and file name. For the gold answer `repo/src/pkg/io.py`, both `src/pkg/io.py` and `pkg/io.py` count, but `io.py` does not. We call the metric normalized exact match (examples in Table D.2).

The path clause raises the baseline's score more than Link's at every length (by 1.9 against 1.3 points at 1M), so without it Link's lead is larger (Section 6.3).

5.3 Splits, Leakage and Eligible Questions

We split the data by task, so no task is shared between splits: 1,097 tasks for training, 2,026 for testing and several hundred more in a separate validation split. Each split spans hundreds of repositories, and task training and all further training draw on the training and validation splits alone. We then drop any test question whose repository appears in training, whose answer equals a training answer, or whose answer appears in the training text as a whole line or, for a path, as a whole path token (Figure D.1). Task training and further training never use the test sessions, which are shared by all compared systems (Table 3). The 2M and 4M sessions arrange the same test tasks as the 1M sessions into longer logs, and at 1M each of the 2,026 test tasks lies in exactly one session.

At 1M we score every eligible question that passes the filters; none is chosen by hand or by a model. At 2M and 4M, to keep the full-attention runs to a manageable time, a generator seeded with the session's name drew 128 eligible questions per session, uniformly without replacement, and all compared systems answer the same 128. The never-seen subset (Figure 3) holds the test tasks that appear nowhere in the manifests of the data used to train or validate Link, the router's own run included. Neither it nor the draw referred to any system's answers.

5.4 Memorization Control

The repositories are real, so a task-trained model might answer from pretraining on public code or from patterns learned in training. We took all 5,394 questions of the 1M test sessions whose answer lies outside Link's exact window: the 5,039 we score and the 355 the leakage filters remove. We removed the log and gave the questions to an earlier Link model trained on the same 1,097 training tasks. Each prompt kept only the log's one-line header and the question block, and decoding was unchanged. The model gave a non-empty answer to every question and answered 11 correctly: 0.2%, with a 95% interval of [0.1, 0.4] from a bootstrap over the 32 sessions. Link's further training used questions on the same training tasks alone, whose answers the leakage filters keep out of the test.

5.5 Statistics

Our 95% intervals on the lead reflect both the variation between training runs and the sampling of test sessions. Both systems answer the same questions on the same sessions, so we pair the comparison by question and session and compute every interval and test within one length. Same-seed runs share a data order, but we treat each system's three training runs as an independent sample rather than pairing them by seed number. Let $\bar{x}$ and $\bar{y}$ be the three-seed means of Link and the baseline, with between-seed variances s_x^2 and s_y^2, and let $\hat{\sigma}_b^2$ be the variance of the seed-averaged paired difference

under a bootstrap over test sessions. The interval is

$$(\bar{x} - \bar{y}) \pm t_{0.975,\nu} \sqrt{\frac{s_x^2 + s_y^2}{3} + \hat{\sigma}_b^2}, \qquad \nu = \frac{\left(s_x^2/3 + s_y^2/3\right)^2}{\frac{(s_x^2/3)^2}{2} + \frac{(s_y^2/3)^2}{2}}, \tag{15}$$

a Welch t-interval with Welch–Satterthwaite degrees of freedom, widened by the session variance (Appendix E.1). With three runs per system its multiplier lies between 2.78 and 4.30, against 1.96 for a normal interval, so the interval is conservative by construction.

Two further checks need no resampling. A two-sided exact sign test on the per-session differences of three-seed means asks whether a session is equally likely to favor either system, taking the six trained runs as given. When all m sessions favor Link, its value is

$$p = 2 \cdot 2^{-m}. \tag{16}$$

It is exact for any number of sessions and does not depend on how questions within a session are correlated. The nine pairings of a Link seed with a baseline seed show whether any pair of runs reverses the result.

6 Results

Table 4 lists what the design of Section 3 aims for and the measurement that bears on each aim. The sections below report the main comparison and then those measurements.

6.1 With Identical Training, Link Is More Accurate Than Attention That Reads Everything

Link is more accurate than full attention at every length, in every session and for every pairing of training runs (Table 5). At 1M tokens, the task-training length of both systems, Link scores 69.5% and the baseline 56.6%, a paired difference of +12.9 [+2.6, +23.1] over 5,039 questions. Link's three-seed mean is higher in all 32 sessions, never by less than 4.3 points (Figure 5a), and all nine pairings of a Link run with a full-attention run favor Link (Figure 5b). The lead stands at all three lengths under the rule of Section 4.2. It is measured at one training budget: Section 6.6 shows that Link was still improving at that budget, and we did not extend the baseline's training, so how the lead changes with longer training is not measured.

6.2 Link Holds Up at Four Times Its Training Length, and Its Lead Widens

Link was trained on 1M-token sessions and keeps most of its accuracy far beyond them. At 4M tokens, four times the training length, it scores 55.5%: 80% of its 1M score, and about what full attention scores at 1M (56.6%). This retention does not depend on the baseline, and the longer training run of Section 6.6 raises it to 92%. Section 6.5 traces it to the router, which selects the answer's chunk as often 3.7M tokens back as nearby.

Against the baseline, the lead widens beyond the training length. At 2M and 4M it is +19.1 [+6.1, +32.1] and +19.2 [+9.0, +29.3] (Table 5, Figure 3), against +12.9 at 1M. At these lengths Link's three-seed mean is higher in every session, by at least 12.2 points at 2M and 13.3 at 4M (Figure 5a). All nine seed pairings favor Link at both lengths (Figure 5b), and in each pairing the lead at 2M and at 4M exceeds that pairing's lead at 1M.

A 4M session holds about four times as many tasks as a 1M session, and the two systems meet that growth differently. At 4M, full attention covers roughly four times as many keys as it saw in task training, with its YaRN factor raised to four times its trained setting (Section 4.2); the comparison does not separate these two changes. Link's attention still reads the prefix, a fixed number of selected chunks and the window, and what grows fourfold is the pool its router chooses from.

Table 4: What the design aims for, and the measurement that bears on each aim.

Design aim	Measurement	Result
Selection should not depend on distance	Share of layers that select the answer's chunk, by distance (Figure 7); accuracy by quarter of answer distance (Figure 6)	Flat out to 3.7M tokens; Link leads in every quarter, by most on the farthest at 2M and 4M
Selecting on the model's own keys beats keyword selection on the text	Identically trained BM25 retrieval on the same base model (Table 7)	Link leads by 18.2 to 21.4 points
The fixed size of the read is not what limits accuracy	Link's lead as the share of the input it reads falls fourfold, from 1M to 4M (Table 5); a longer training run with the same read (Table 8)	The lead grows from +12.9 to +19.2 points; 77.0% at 4M with the same read (one run)
Cost and GPU memory stay bounded	Measured read time and peak GPU memory (Table 9)	Time exponent 1.15 against 1.93 from 2M to 4M; 23.9 to 24.6 GiB

Table 5: Normalized exact match (%) on held-out test sessions, three-seed means, after identical task training. Difference is Link minus GQA + YaRN, computed from unrounded scores; its 95% interval accounts for both training runs and test sessions (Section 5.5). Table 6 extends the comparison to every subset and both scoring rules.

Length	Sessions	Questions	Link	GQA + YaRN	Difference [95% CI]	Sessions led by Link	Sign test (two-sided)
1M	32	5,039	**69.5**	56.6	+12.9 [+2.6, +23.1]	32 of 32	$p < 10^{-9}$
2M	16	2,048	**65.4**	46.3	+19.1 [+6.1, +32.1]	16 of 16	$p < 10^{-4}$
4M	8	1,024	**55.5**	36.3	+19.2 [+9.0, +29.3]	8 of 8	$p = 0.008$

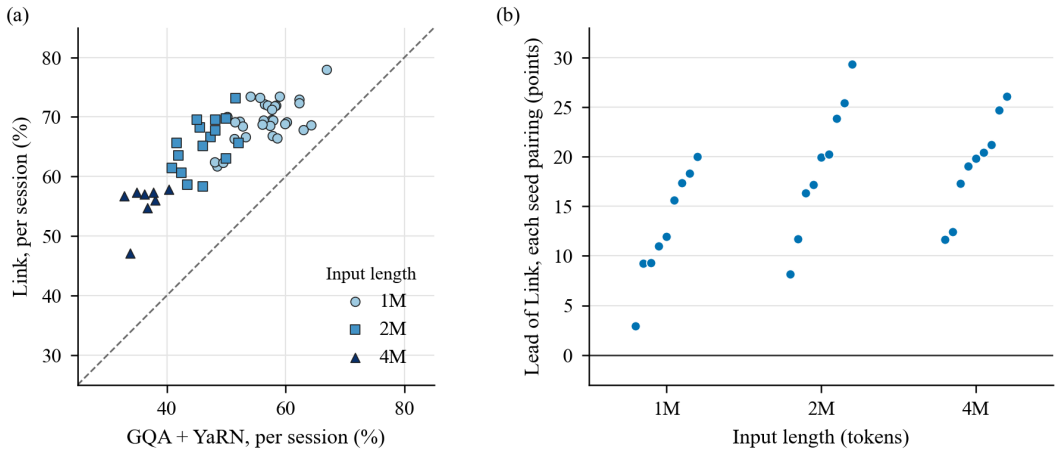


Figure 5: (a) Per-session scores at 1M, 2M and 4M tokens, three-seed means. Each point is one of the 56 test sessions; points above the dashed diagonal mark sessions where Link scores higher. (b) Link's lead at each length for the nine pairings of a Link run with a GQA + YaRN run (27 in all), computed over all scored questions.

Table 6: Link's lead over GQA + YaRN in points on every subset and under both scoring rules, three-seed means after identical task training, with 95% intervals by (15) and the number of questions.

Questions	1M	2M	4M
All scored	+12.9 [+2.6, +23.1]; 5,039	+19.1 [+6.1, +32.1]; 2,048	+19.2 [+9.0, +29.3]; 1,024
Answer beyond the native window	+13.4 [+3.0, +23.8]; 3,744	+19.4 [+6.1, +32.6]; 1,823	+19.5 [+10.0, +29.0]; 950
Never-seen tasks	+12.4 [+1.8, +23.0]; 2,285	+17.1 [+4.3, +29.9]; 896	+18.1 [+4.0, +32.2]; 478
Stricter never-seen subset	+13.3; 2,042	+18.6; 801	+18.5; 430
All scored, strict exact match	+13.4	+19.7	+19.4
Sessions led by Link (smallest lead)	32 of 32 (4.3)	16 of 16 (12.2)	8 of 8 (13.3)
Seed pairings favoring Link	9 of 9	9 of 9	9 of 9
Two-sided sign test	$p < 10^{-9}$	$p < 10^{-4}$	$p = 0.008$

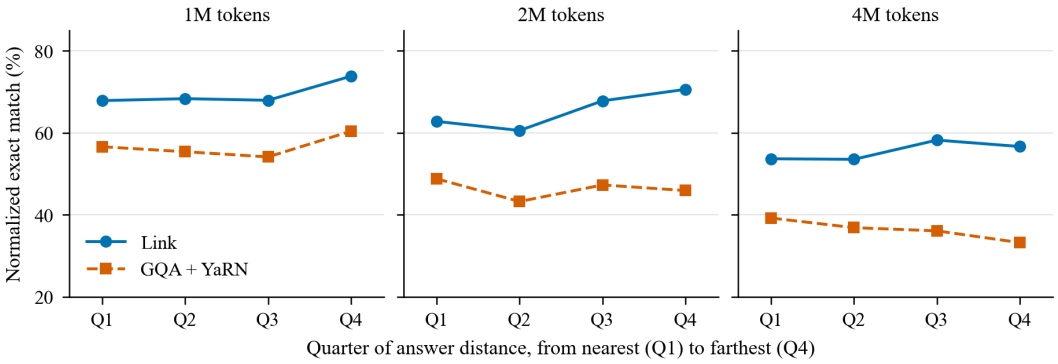


Figure 6: Normalized exact match by quarter of answer distance, from nearest to farthest, three-seed means after identical task training. We form the quarters separately at each length; a quarter holds 1,259 or 1,260 questions at 1M, 512 at 2M and 256 at 4M.

6.3 The Lead Holds Under Every Stricter Test

The lead holds on every subset on which it could be expected to weaken, and under the stricter scoring rule (Table 6).

Beyond the native window. Most scored questions have their answer more than 262,144 tokens back, beyond Qwen3.5-9B's native window: 3,744 of 5,039 at 1M, 1,823 of 2,048 at 2M and 950 of 1,024 at 4M (Figure D.3). The baseline reaches these answers only through YaRN's rescaling. On these questions Link leads by +13.4 [+3.0, +23.8] at 1M, +19.4 [+6.1, +32.6] at 2M and +19.5 [+10.0, +29.0] at 4M (95% intervals as for Table 5), each at least as large as its overall lead at that length.

By answer distance. We split each length's questions into four equal parts by answer distance (Figure 6). Link leads in every quarter at all three lengths. At 2M and 4M the lead rises from about 14 points on the nearest quarter to +24.7 and +23.4 on the farthest, while the baseline does worse on the farthest quarter than on the nearest.

Never-seen tasks. On the never-seen tasks (Section 5.3), Link leads by +12.4 [+1.8, +23.0] at 1M (2,285 questions), +17.1 [+4.3, +29.9] at 2M (896) and +18.1 [+4.0, +32.2] at 4M (478). Each Link training run again scores above each baseline run, and Link's three-seed mean is higher in every session, at all three lengths. A stricter subset, which also requires the titled task named by a next-task question to be never-seen, gives a larger lead at every length: +13.3 at 1M (2,042 questions), +18.6 at 2M (801) and +18.5 at 4M (430).

Strict exact match. Under strict exact match after the same normalization, without the path clause (Section 5.2), Link's lead over all scored questions is +13.4 points at 1M, +19.7 at 2M and +19.4 at 4M (Table 6).

Table 7: Normalized exact match (%) on the test sessions of Table 5, three-seed means, after identical task training. Difference is Link minus keyword retrieval (BM25), computed from unrounded scores, with 95% intervals as for Table 5.

Length	Sessions	Link	Keyword retrieval (BM25)	Difference [95% CI]	Sessions led by Link
1M	32	**69.5**	49.5	+19.9 [+10.0, +29.9]	32 of 32
2M	16	**65.4**	44.0	+21.4 [+8.2, +34.5]	16 of 16
4M	8	**55.5**	37.3	+18.2 [+7.9, +28.4]	8 of 8

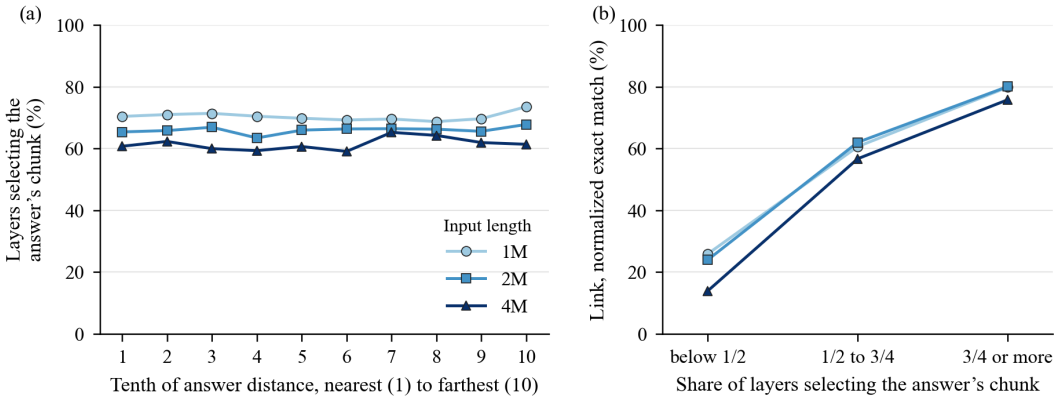


Figure 7: What the router selects at the last token of each scored question, averaged over the eight attention layers and over the three task-trained models. (a) Share of layers whose selected chunks include one that holds the gold answer, by tenth of answer distance; the farthest tenths lie at median distances of 0.94M, 1.88M and 3.70M tokens. (b) Link's normalized exact match by that share. The three bins hold 417, 1,553 and 3,069 questions at 1M, 307, 707 and 1,034 at 2M, and 217, 384 and 423 at 4M.

Question types. Link scores above GQA + YaRN on 20 of the 21 combinations of question type and length; the exception is the path of a next task at 1M, 59.9 against 60.5 (Figure F.1).

6.4 Selecting Inside the Model Beats Keyword Retrieval Outside It

Link leads identically trained keyword retrieval by about 20 points at every length (Table 7). Its selection lies between the model's own attention, which reads every position, and retrieval, which selects passages from the raw text outside the model, so retrieval is the natural second comparison. The retrieval baseline is the same base model reading 98,304 tokens of the log for each question: its last 32,768 tokens and the 65,536 tokens of passages that BM25 ranks highest for the question. This budget did best of three sizes on validation data and was fixed before any test run. This baseline receives the same task training, on the same sessions, with three seeds. Link's lead is +19.9 [+10.0, +29.9] points at 1M, +21.4 [+8.2, +34.5] at 2M and +18.2 [+7.9, +28.4] at 4M. Its three-seed mean is higher in every test session at all three lengths, and all nine pairings of training runs favor Link at each length.

6.5 The Router Finds the Answer as Often 3.7M Tokens Back as Nearby

Section 3.3 designed the router to judge a chunk by what it contains, wherever it lies. Our evaluation records let us check this directly: for every scored question they note whether, at the question's last token, the chunks that each attention layer selects include one that holds the gold answer. Figure 7 averages this record over the eight attention layers and over the three task-trained models.

Within each length the share hardly depends on how far back the answer lies: it stays between 69% and 74% across the ten tenths at 1M, between 63% and 68% at 2M, and between 59% and 65% at 4M, where the farthest tenth lies 3.7M tokens back, almost four times farther than any distance in task

Table 8: Normalized exact match (%) on the test sessions of Table 5. In parentheses is each score's lead in points over GQA + YaRN, which receives task training only; leads are computed before rounding. All columns are three-seed means except extended task training, which is one run.

Length	GQA + YaRN	Link, task training	Link + reinforcement learning	Link + extended task training (one run)
1M	56.6	69.5 (+12.9)	77.8 (+21.2)	84.0
2M	46.3	65.4 (+19.1)	72.9 (+26.7)	82.7
4M	36.3	55.5 (+19.2)	66.3 (+30.0)	77.0

training. Its mean falls from 70.4% at 1M to 66.0% at 2M and 61.5% at 4M, as the store grows to four times as many chunks. Accuracy follows the share. When fewer than half of the layers select the answer's chunk, Link answers 25.8%, 23.9% and 13.8% of the questions at 1M, 2M and 4M; when three quarters or more do, it answers 79.9%, 80.1% and 75.8%.

6.6 Further Training Raises Link Without Enlarging Its Read

Further training raises Link's scores without enlarging its read (Table 8). The baseline keeps its task-trained weights throughout the table, so these leads combine the effect of the read with that of the added training; Table 5 remains the controlled comparison.

Reinforcement learning. At each query the router makes a discrete, set-valued decision. By (10) the language-model loss carries no gradient for the chunks it leaves out, and whether the answer is right is not a differentiable function of the selection. Policy-gradient methods fit this setting, because they improve a stochastic policy from sampled outcomes and their rewards.

In this stage the model acts as a stochastic policy. For a training question x, a rollout τ consists of the read's chunk selections and the tokens of the answer, with

$$\log \pi_{\phi,\theta}(\tau \mid x) = \sum_t \log \pi_\phi(S_t \mid q_t) + \sum_i \log \pi_\theta(y_i \mid y_{<i}, R), \tag{17}$$

where ϕ parameterizes the read and θ the weights that write the answer. We maximize the expected reward

$$J(\phi,\theta) = \mathbb{E}_x \, \mathbb{E}_{\tau \sim \pi_{\phi,\theta}(\cdot|x)} \, R(\tau) \tag{18}$$

and, for K sampled rollouts per question, estimate its gradient with a leave-one-out baseline:

$$\hat{g} = \frac{1}{K} \sum_{a=1}^{K} \left(R(\tau_a) - \frac{1}{K-1} \sum_{b \neq a} R(\tau_b) \right) \nabla_{\phi,\theta} \log \pi_{\phi,\theta}(\tau_a \mid x). \tag{19}$$

The estimator requires no gradient through the selection, which resolves the problem posed by (10); Appendix C.2 shows that it is unbiased. The reward is computed automatically, with no learned judge, and correctness under normalized exact match dominates it: any rollout with a correct answer earns more than any rollout without one. Validation and testing use the trained policy's greedy read, which at the start of the stage is exactly the deterministic read of the task-trained model. Before a run's first update, we checked on a sample of validation questions that its greedy read reproduces its starting model's answers token for token.

Each of three runs starts from one of the three task-trained Link models with its own sampling seed. Training questions come from the 1M-token training sessions and validation questions from the validation split, so the test tasks, the leakage filters and the never-seen subset apply unchanged. Stopping and checkpoint choices used validation data alone and were fixed before any run was tested (Table D.3): each run is tested once, at the checkpoint with the highest validation exact match on a fixed grid, ties going to the earlier (Figure C.1).

Reinforcement learning raises Link above its starting models by +8.3 [+7.7, +9.1] points at 1M, +7.5 [+6.6, +8.6] at 2M and +10.8 [+8.9, +12.6] at 4M; these intervals resample test sessions and take

the trained runs as given. Against grouped-query attention with YaRN, which received task training only, the lead becomes +21.2 [+11.3, +31.1] at 1M, +26.7 [+16.9, +36.4] at 2M and +30.0 [+22.7, +37.3] at 4M (95% intervals as for Table 5), and its three-seed mean is higher in every test session.

Extended task training. This run asks whether the fixed size of Link's read limits its accuracy. It keeps the read unchanged, with the same number of chunks per query, and continues the task training of the seed-0 model on the same training sessions, with as many GPUs and the same time limit as one reinforcement-learning run. The same validation rule chose its checkpoint before testing. It reaches 84.0%, 82.7% and 77.0% at 1M, 2M and 4M, keeping 92% of its 1M score at 4M, against 80% for the task-trained models. It is a single run of Link alone, so it carries no interval over training runs. That this one run gains more than reinforcement learning at the same compute (77.0% against a three-run mean of 66.3% at 4M) shows that the task-trained Link models were far from converged.

7 Efficiency

We timed the read of every test session for the seed-0 models of both systems at the end of task training, on H100 80GB GPUs at batch size 1 (Table 9). At every length, Link's slowest session read is faster than the fastest full-attention read. At 4M, the dense read occupies three GPUs for 57 minutes, while Link needs one GPU for under six. Reinforcement learning leaves the size of the read unchanged: every Link checkpoint we test reads the same number of chunks per query as the models we timed.

A read is every forward pass over the session, from its first token to its last, timed with the GPUs synchronized before and after; model loading, tokenization and answering are excluded. For Link the read includes the router and, from 2M on, every transfer between the host-memory store and the GPU. The two systems share one code base, numerical precision and software stack, use the same kernels for each kind of layer, and pass each session through the model identically (Appendix B.4).

When the baseline's cache is too large for one GPU, it splits the model's layers across GPUs: two at 2M and three at 4M, a count set for memory headroom (Table D.3). The GPUs run their layers in turn, so the split makes room for the cache but does not shorten the read; at 1M, a read split over two GPUs took as long as a read on one. Its times are therefore those of one GPU doing all of its work, and each speedup compares one GPU's work with one GPU's work.

At 1M tokens Link keeps its store on the GPU, and its peak memory (50.9 GiB) matches that of full attention (50.7 GiB): at this length Link saves time, not memory. From 2M on, the store lives in host memory. Link's GPU peak is then 23.9 GiB and barely moves when the input doubles to 4M (24.6 GiB), while the store doubles with the input, from about 60 GiB of host memory at 2M to about 120 GiB at 4M: the keys and values that full attention holds on its GPUs. Between 2M and 4M, Link's read time roughly doubles and that of full attention grows almost fourfold (Figure 2). Writing

Table 9: Median session-read time and memory for the timed seed-0 models, over 32, 16 and 8 sessions. Speedup is the ratio of the unrounded medians (61.4 and 241.5 s at 1M, 155.6 and 890.5 s at 2M, 344.6 and 3,397.8 s at 4M). GPU memory is given in GiB (2^{30} bytes). Peak GPU memory is the peak allocated memory on the busiest GPU over a session's read and its answers, median over sessions; each Link entry also says where its store is kept.

Length	Link read time	GQA + YaRN read time	Wall-clock speedup	Link peak GPU memory	GQA + YaRN peak GPU memory
1M	61 s, 1 GPU	241 s, 1 GPU	3.9×	50.9 GiB, store on GPU	50.7 GiB (1 GPU)
2M	156 s, 1 GPU	891 s, 2 GPUs	5.7×	23.9 GiB, store in host memory	43.1 GiB on the busiest of 2 GPUs
4M	345 s, 1 GPU	3,398 s, 3 GPUs	9.9×	24.6 GiB, store in host memory	60.2 GiB on the busiest of 3 GPUs

$T(n)$ for the median read time, the local scaling exponent

$$\gamma = \log_2 \frac{T(2n)}{T(n)} \tag{20}$$

is 1.34 from 1M to 2M and 1.15 from 2M to 4M for Link, against 1.88 and 1.93 for the dense read, in line with the $\Theta(n^2)$ of (2); Figure B.1 plots the read time per million tokens. The medians of each session's own ratio of read times, 4.0×, 5.7× and 9.9×, agree with the ratios of the medians. Timed the same way, each median for the task-trained seed-1 and seed-2 models of both systems lies within 3% of its seed-0 value, except one of Link's at 4M, which is 11% lower.

Splitting the baseline's sequence across GPUs rather than its layers (Liu et al., 2024a) could shorten its read by at most the number of GPUs, but not its total work. However it is split, a dense read takes at least 9.9× the GPU time of Link's at 4M, 5.7× at 2M and 3.9× at 1M, while Link reads on one GPU (Appendix B.4).

8 Link Inside Complete Systems

The coding-log benchmark isolates the read. To see how Link behaves inside complete question-answering systems, on other kinds of text and on questions that need reasoning as well as recall, we built one system for each of two public benchmarks and scored each on the official test set with the official scorer. Each system pairs Link's learned read with BM25 as a complementary source of evidence: Link selects from key-value states computed over the whole input, and BM25 matches exact terms; in both systems Link ranks passages that the model then reads again as text. The BABILong system is trained on its task, and the LongBench v2 system uses the router that every Link model in the coding-log comparison starts from. Sections 6.1 and 6.4 measure the learned read under controlled conditions; the scores below are full-system results, which do not isolate the read's share.

8.1 BABILong

BABILong (Kuratov et al., 2024) puts the facts of bAbI reasoning stories (Weston et al., 2016) into long background text and asks a question that needs them. We use qa1 to qa5 of the official test split, 100 questions per task and length, and the benchmark's own answer-matching scorer. As its leaderboard does, we average the five tasks at each length.

Our BABILong system is Qwen3.5-9B with Link, trained on the task. On longer inputs it reads a second time before it answers: the passages Link ranks highest for the question, passages found by keyword search, and the end of the input.

At 512K and 1M tokens, about two and four times the model's native window, the system scores 68.6% and 66.6%. Our control is Qwen3.5-9B trained on the same questions for the same number of steps, answering from keyword-retrieved excerpts; the Link system leads it by +8.0 [+3.8, +12.2] and +6.2 [+2.0, +10.4] points (Table 10). The lead comes from qa2, whose answers need two supporting facts: there the Link system answers 56% and 55% of the questions, against 16% and 19%, while the two systems are about level on the other four tasks taken together (Figure 8). Each system's way of reading was chosen on validation questions, Link's per task and the control's once for all tasks. Appendix G.1 gives the full set-up, and reports a later version of the system, with short fact traces, on inputs up to 128K.

8.2 LongBench v2

LongBench v2 (Bai et al., 2025) has 503 multiple-choice questions over documents, dialogue histories, code repositories and structured data, with contexts from about 8K to 2M words. Where the coding logs ask for one line, these questions ask for reasoning over the whole input. We ran all 503 with the benchmark's zero-shot chain-of-thought prompt, its answer extraction and its scorer.

For LongBench v2, Link sits at the front of a complete long-document question-answering system built on Qwen3.5-9B. Link reads the whole input and ranks its passages for the question, with the

Table 10: BABILong at 512K and 1M tokens, official test split, average over qa1 to qa5 (%). The difference compares the Link system with Qwen3.5-9B trained on the same training questions for the same number of steps and answering from keyword-retrieved excerpts, the reading it did best on validation; it is paired by question. Untrained Qwen3.5-9B, answering from keyword-retrieved excerpts of the same inputs, is shown for reference. Intervals resample questions within each task and length.

Length	Questions	Link system [95% CI]	Qwen3.5-9B trained the same way, keyword excerpts	Difference [95% CI]	Untrained Qwen3.5-9B, keyword excerpts
512K	500	**68.6** [65.0, 72.0]	60.6	+8.0 [+3.8, +12.2]	37.0
1M	500	**66.6** [63.2, 70.0]	60.4	+6.2 [+2.0, +10.4]	33.8

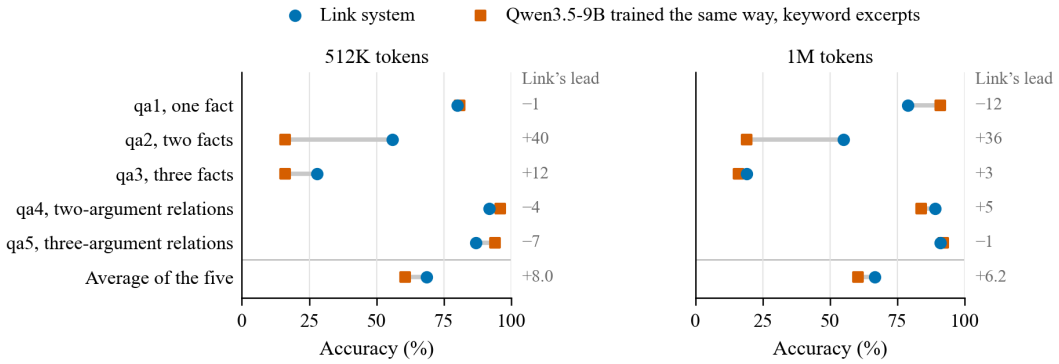


Figure 8: BABILong accuracy by task at 512K and 1M tokens, on the official test split with the official scorer, 100 questions per point: the Link system and Qwen3.5-9B trained on the same questions for the same number of steps and answering from keyword-retrieved excerpts. Tasks qa1 to qa3 need one, two and three supporting facts, and qa4 and qa5 ask about two- and three-argument relations. Table G.2 gives the numbers.

router that every Link model above starts from. The model, in thinking mode, then answers from its native window, which holds the passages that Link and keyword search select together with as much of the original input as fits, and it decodes with self-consistency (Wang et al., 2023). It is the second version of the system that we scored on this benchmark; its configuration was chosen on separate validation data, and Appendix G.2 describes it in full.

The system scores 61.2% on all 503 questions (95% CI [56.9, 65.4]): 70.3 on easy and 55.6 on hard questions, and 65.0, 59.5 and 58.3 on short, medium and long inputs. The public leaderboard (THUDM, 2025) lists single models that answer once, so it gives scale rather than a like-for-like comparison with a complete system: Gemini-2.5-Pro scores 63.3 there, Gemini-2.5-Flash 62.1 and Qwen3-235B-A22B-Thinking-2507 60.6, and human experts 53.7.

9 Discussion

Reading everything is not the same as using everything. Grouped-query attention shrinks the key-value cache and YaRN extends attention past the trained window, but neither changes what each query reads, which is precisely what Link learns. As the input grows from 1M to 4M tokens, the share of the history each Link query reads, B/n, falls by a factor of four, from about 0.6% to 0.16%. Link nonetheless keeps 80% of its 1M score at 4M (55.5 against 69.5), whereas the baseline, which reads every position, keeps 64% (36.3 against 56.6). At 2M and 4M the baseline also does worse on the farthest answers than on the nearest, while Link's router selects the answer's chunk about as often at every distance (Section 6.5).

Attending to the entire input, then, does not by itself make the evidence in it usable. Our interpretation is that at these lengths the difficulty lies mainly in locating the few positions that matter, and a selection trained on the model's own attention addresses that problem directly. Figure 7 points

the same way: when three quarters or more of the layers select the answer's chunk, Link answers 76–80% of those questions at every length, and most of its errors, from 60% at 1M to 78% at 4M, fall on questions where fewer layers do.

The comparison with keyword retrieval supports the same interpretation. The BM25 baseline also gives attention only a small part of the log and receives identical training, yet it trails Link by about 20 points at every length. What distinguishes the two is the basis of selection: lexical overlap with the question, applied to the raw text before the model runs, in one case; in the other, keys the model has computed over the whole input. At 4M the dense read (36.3%) scores no higher than keyword retrieval (37.3%), though it runs there with a YaRN factor its training never used. The extended training run indicates, finally, that the size of the read was not what limited Link at the end of task training: with the same number of chunks per query, further task training reaches 77.0% at 4M tokens.

10 Conclusion

Link is a learned sparse read that lets the attention layers of Qwen3.5-9B select, for each query, chunks from a key-value store that can leave the GPU. Trained identically to the model's own grouped-query attention with YaRN, and tested on tasks and repositories held out from task training, it recalls verbatim details from long coding sessions more accurately at 1M, 2M and 4M tokens, the last two beyond the length either system was trained on, and further training of Link, by reinforcement learning and in a single longer run, raises it without enlarging the read. It reads a 4M-token session in a median of under six minutes on one GPU, using at least 9.9× less GPU time than full attention, which takes close to an hour with its cache spread over three GPUs. Systems that use it to rank passages lead a matched control on BABILong beyond the native window and score 61.2% on LongBench v2 (Section 8).

Because each query reads a selection of fixed size, an agent's history can grow past both the native window and the memory of the GPU without raising the attention cost of a query. Agents that work for weeks need exactly this: to act on a detail seen millions of tokens earlier, at an attention cost that does not grow with everything that came before it.

This technical report presents our first results with Link. A fuller paper is in preparation.

References

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. (2023). GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of EMNLP*.

Bai, Y., Tu, S., Zhang, J., et al. (2025). LongBench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of ACL*.

Bertsch, A., Alon, U., Neubig, G., and Gormley, M. R. (2023). Unlimiformer: Long-range transformers with unlimited length input. In *Advances in NeurIPS*.

Chen, S., Wong, S., Chen, L., and Tian, Y. (2023). Extending context window of large language models via positional interpolation. arXiv:2306.15595.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. (2019). Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of ACL*.

DeepSeek-AI (2025). DeepSeek-V3.2: Pushing the frontier of open large language models. arXiv:2512.02556.

Gao, Y., Zeng, Z., Du, D., et al. (2024). SeerAttention: Learning intrinsic sparse attention in your LLMs. arXiv:2410.13276.

Gu, A., and Dao, T. (2024). Mamba: Linear-time sequence modeling with selective state spaces. In *Proceedings of COLM*.

Hsieh, C.-P., Sun, S., Kriman, S., et al. (2024). RULER: What's the real context size of your long-context language models? In *Proceedings of COLM*.

Jiang, H., Li, Y., Zhang, C., et al. (2024). MInference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention. In *Advances in NeurIPS*.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. (2020). Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proceedings of ICML*.

Kuratov, Y., Bulatov, A., Anokhin, P., Rodkin, I., Sorokin, D., Sorokin, A., and Burtsev, M. (2024). BABILong: Testing the limits of LLMs with long context reasoning-in-a-haystack. In *Advances in NeurIPS (Datasets and Benchmarks Track)*.

Lee, H., Park, G., Suh, J., and Hwang, S. J. (2025). InfiniteHiP: Extending language model context up to 3 million tokens on a single GPU. arXiv:2502.08910.

Lewis, P., Perez, E., Piktus, A., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in NeurIPS*.

Li, Y., Huang, Y., Yang, B., et al. (2024). SnapKV: LLM knows what you are looking for before generation. In *Advances in NeurIPS*.

Liu, D., Chen, M., Lu, B., et al. (2025). RetrievalAttention: Accelerating long-context LLM inference via vector retrieval. In *Advances in NeurIPS*.

Liu, H., Zaharia, M., and Abbeel, P. (2024a). Ring attention with blockwise transformers for near-infinite context. In *Proceedings of ICLR*.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. (2024b). Lost in the middle: How language models use long contexts. *Transactions of the ACL*, 12.

Lu, E., Jiang, Z., Liu, J., et al. (2025). MoBA: Mixture of block attention for long-context LLMs. In *Advances in NeurIPS*.

Mohtashami, A., and Jaggi, M. (2023). Random-access infinite context length for transformers. In *Advances in NeurIPS*.

Pang, R. Y., Parrish, A., Joshi, N., Nangia, N., Phang, J., Chen, A., Padmakumar, V., Ma, J., Thompson, J., He, H., and Bowman, S. R. (2022). QuALITY: Question answering with long input texts, yes! In *Proceedings of NAACL*.

Peng, B., Quesnelle, J., Fan, H., and Shippole, E. (2024). YaRN: Efficient context window extension of large language models. In *Proceedings of ICLR*.

Qwen Team (2026). Qwen3.5-9B model card. Hugging Face.

Robertson, S., and Zaragoza, H. (2009). The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4), 333–389.

Satterthwaite, F. E. (1946). An approximate distribution of estimates of variance components. *Biometrics Bulletin*, 2(6), 110–114.

Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., and Liu, Y. (2021). RoFormer: Enhanced transformer with rotary position embedding. arXiv:2104.09864.

Sun, H., Chang, L.-W., Bao, W., et al. (2025). ShadowKV: KV cache in shadows for high-throughput long-context LLM inference. In *Proceedings of ICML*.

Tang, J., Zhao, Y., Zhu, K., Xiao, G., Kasikci, B., and Han, S. (2024). Quest: Query-aware sparsity for efficient long-context LLM inference. In *Proceedings of ICML*.

THUDM (2025). LongBench v2 leaderboard. https://longbench2.github.io.

Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention is all you need. In *Advances in NeurIPS*.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. (2023). Self-consistency improves chain of thought reasoning in language models. In *Proceedings of ICLR*.

Welch, B. L. (1947). The generalization of "Student's" problem when several different population variances are involved. *Biometrika*, 34(1–2), 28–35.

Weston, J., Bordes, A., Chopra, S., Rush, A. M., van Merriënboer, B., Joulin, A., and Mikolov, T. (2016). Towards AI-complete question answering: A set of prerequisite toy tasks. In *Proceedings of ICLR*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8, 229–256.

Wu, Y., Rabe, M. N., Hutchins, D., and Szegedy, C. (2022). Memorizing transformers. In *Proceedings of ICLR*.

Xiao, C., Zhang, P., Han, X., et al. (2024a). InfLLM: Training-free long-context extrapolation for LLMs with an efficient context memory. In *Advances in NeurIPS*.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. (2024b). Efficient streaming language models with attention sinks. In *Proceedings of ICLR*.

Xu, Z., Ye, J., Liu, X., et al. (2024). DetectiveQA: Evaluating long-context reasoning on detective novels. arXiv:2409.02465.

Yang, S., Kautz, J., and Hatamizadeh, A. (2025). Gated delta networks: Improving Mamba2 with delta rule. In *Proceedings of ICLR*.

Yuan, J., Gao, H., Dai, D., et al. (2025). Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of ACL*.

Zhang, Z., Sheng, Y., Zhou, T., et al. (2023). H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in NeurIPS*.

Appendix

The appendices give the procedures, derivations, diagnostics and complete results behind the main text. Equations numbered (1) to (20) are those of the main text; equations, figures, tables and algorithms numbered here carry the letter of their appendix.

Appendix A. Terms and Notation

A.1 Terms

Table A.1 defines, in alphabetical order, the technical terms that the paper uses without defining them in place, with the section or equation where each first matters.

Table A.1: Terms used in the paper.

Term	Meaning
bAbI	Synthetic question-answering tasks built from short stories about people, objects and places (Weston et al., 2016).
BABILong	Benchmark that hides the supporting facts of bAbI stories in long background text from books and asks a question that needs them; tasks qa1 to qa5 need one, two or three facts, or two- and three-argument relations (Kuratov et al., 2024).
BM25	Lexical ranking function that scores a passage by the query terms it contains, with term-frequency saturation, inverse document frequency and passage-length normalization (Robertson and Zaragoza, 2009).
Bootstrap over sessions	Estimate of a statistic's sampling variability from its values on test sets drawn with replacement from the test sessions of one length (Section 5.5).
Chunk store	The keys and values of the positions outside the exact window, kept as consecutive chunks of L positions in GPU or host memory (Section 3.2).
Exact window	The prefix P and the recent window W_t, which every query reads in full (Section 3.2).
Gated DeltaNet	Linear-time recurrent layer whose fixed-size matrix state is updated at each token by a gated delta rule: the state decays by a learned gate and is corrected toward the new key–value association (Yang et al., 2025).
Grouped-query attention (GQA)	Attention in which each group of $G = H/H_k$ query heads shares one key-value head, which shrinks the key-value cache by a factor of G (Ainslie et al., 2023).
Key-value cache	The keys and values of all earlier positions that an attention layer keeps so that later queries can attend to them without recomputation; it grows linearly with the input, (3).
KL divergence	$\mathrm{KL}(p \,\|\, q) = \sum_j p(j) \log \big(p(j)/q(j)\big)$, which is zero only when the two distributions agree, (14).
Leave-one-out baseline	The mean reward of the other $K - 1$ rollouts of the same question, subtracted from each rollout's reward, (19).
LongBench v2	Benchmark of 503 multiple-choice questions over long documents, dialogue histories, code repositories and structured data, with contexts of about 8K to 2M words (Bai et al., 2025).
Native window	The longest input a model was trained to attend over with its original position encoding: 262,144 tokens for Qwen3.5-9B.
Normalized exact match	The scoring rule of Section 5.2: the normalized first line of the reply must equal the gold answer, and a path may also match in a form that keeps at least its last directory and file name.
Policy gradient	An estimate of the gradient of an expected reward from sampled outcomes, as the reward-weighted gradient of each sample's log-probability (Williams, 1992).

Table A.1, continued

Term	Meaning
Qwen3.5-9B	The base model: a 9B-parameter hybrid that interleaves Gated DeltaNet layers with eight softmax-attention layers and has a native window of 262,144 tokens (Qwen Team, 2026).
Rotary position encoding	Encoding that rotates pairs of query and key coordinates by angles proportional to position, so that their inner product depends on relative position (Su et al., 2021); Qwen3.5 applies it to part of each head's coordinates.
Router	The small learned network that scores the stored chunks for each query and key-value head and keeps the k highest, (7) to (9).
Self-consistency	Decoding that samples several reasoning chains and returns the answer they most often reach (Wang et al., 2023).
Self-distillation	Training the router toward a target computed from the attention of the unmodified model itself, (13) and (14).
Sign test	Exact test of whether positive and negative differences are equally likely, from the counts of each, (16).
Thinking mode	Qwen3.5's mode that writes a reasoning trace before the answer; it is off on the coding-log benchmark and on in the LongBench v2 system.
Welch t-interval	Confidence interval for a difference of two means with unequal variances, with Welch–Satterthwaite degrees of freedom (Welch, 1947; Satterthwaite, 1946), (15).
YaRN	Method that extends a rotary-position model past its native window. It interpolates the rotary frequencies dimension by dimension, leaving the fastest-rotating dimensions nearly unchanged and stretching the slowest by the scale factor $s = N/N_0$, and it rescales the attention logits by a temperature that depends on s (Peng et al., 2024).

A.2 Notation

Table A.2 lists the symbols of Sections 3 to 7 in order of first use. Symbols that appear only in these appendices are defined where they first occur, and the letter m, which has two meanings, is disambiguated by context.

Table A.2: Notation of Sections 3 to 7.

Symbol	Meaning	Defined in
H	Query heads in an attention layer	Section 3.1
H_k	Key-value heads in an attention layer	Section 3.1
d	Dimension of one head	Section 3.1
t	Position of the query	(1)
q_t	Query at position t	(1)
k_i, v_i	Key and value at position i	(1)
$\alpha_{t,i}$	Attention weight of query t on position i	(1)
a_t	Output of one attention head at position t	(1)
n	Length of the session, in tokens	(2)
$C_{\text{full}}(n)$	Key-query inner products per query head for a full read	(2)
$M_{\text{full}}(n)$	Entries in one layer's key-value cache under full attention	(3)
N_0	Native window of Qwen3.5-9B, 262,144 tokens	(4)
N	Target length for YaRN	(4)
s	YaRN scale factor, N/N_0	(4)
P	Prefix at the start of the input, always read in full	(5)

Table A.2, continued

Symbol	Meaning	Defined in
W_t	Recent window of the latest positions ending at t, always read in full	(5)
L	Length of every chunk	(5)
C_j	The j-th chunk of the store	(5)
S_t	Chunks selected for query t	(5)
R_t	Positions that query t reads	(5)
k	Chunks selected per query and key-value head	(6)
B	Bound on the read, $\lvert P\rvert + kL + \lvert W_t\rvert$	(6)
G	Query heads that share one key-value head, H/H_k	Section 3.3
ϕ	The router's parameters	Section 3.3
$\tilde{k}$, $\tilde{q}$	Keys and queries restricted to their position-free coordinates	(7), (8)
$\tilde{K}_{h,j}$	Position-free keys of chunk j in key-value head h	(7)
$\psi(\tilde{K})$	Chunk statistics: the mean key and the coordinatewise maximum and minimum	(7)
$s_{h,j}$	The router's summary of chunk j in head h	(7)
W_h^K, b_h^K	The router's projection of a chunk's statistics	(7)
r	Dimension of the router's scoring space	(8)
W_h^Q, b_h^Q	The router's projection of a group's queries	(8)
$u_{h,t}$	The router's summary of the G queries at position t in head h	(8)
$\sigma_{t,h}(j)$	Router score of chunk j for query t in head h	(9)
$S_{t,h}$	The k highest-scoring chunks for query t in head h	(9)
$C_{\text{attn}}(n)$	Key-query inner products per query head for Link's read	(11)
m	Number of stored chunks, $\lfloor (n - \lvert P\rvert - \lvert W_t\rvert)/L \rfloor$	Section 3.4
$M_{\text{GPU}}^{\text{kv}}(n)$, $M_{\text{host}}^{\text{kv}}(n)$	Key-value memory on the GPU and in host memory	(12)
$p_{t,h}(j)$	Share of full attention's mass on chunk j: the router's target	(13)
$\alpha_{t,i}^{(g)}$	Attention weight of query head g	(13)
$\mathcal{L}(\phi)$	The router's distillation loss	(14)
$w_{t,h}$	Weight of a query in that loss, proportional to its attention mass on the store	(14)
$\bar{x}$, $\bar{y}$	Three-seed means of Link and the baseline	(15)
s_x^2, s_y^2	Between-seed variances of Link and the baseline	(15)
$\hat{\sigma}_b^2$	Variance of the seed-averaged paired difference under a bootstrap over sessions	(15)
ν	Welch–Satterthwaite degrees of freedom	(15)
$t_{0.975,\nu}$	The 0.975 quantile of Student's t with ν degrees of freedom	(15)
m	Number of test sessions, in the sign test	(16)
p	Two-sided exact sign-test value	(16)
x	A training question	(17)
τ	A rollout: the read's selections and the tokens of the answer	(17)
θ	The weights that write the answer	(17)
$\pi_{\phi,\theta}$	The policy over rollouts	(17)
$R(\cdot)$	Reward of a rollout	(18)
J	Expected reward	(18)
K	Rollouts sampled per question	(19)
$\hat{g}$	Gradient estimate with a leave-one-out baseline	(19)
$T(n)$	Median time to read a session of length n	(20)
γ	Local scaling exponent of read time, $\log_2 T(2n)/T(n)$	(20)

Appendix B. The Read in Full

Algorithms B.1 and B.2 restate the read of Section 3.2 and the store of Section 3.5 for a single attention layer, marking each step with the equation it implements.

Algorithm B.1 The read of one query at position t, in one attention layer

Input queries $q_{t,h,g}$ for key-value heads $h = 1, \dots, H_k$ and query heads $g = 1, \dots, G$; the prefix P and the recent window W_t; the stored chunks $C_1, \dots, C_m$ and their summaries $s_{h,j}$
for each key-value head h **do**
  $u_{h,t} \leftarrow (1/\sqrt{r})\big(W_h^Q[\tilde{q}_{t,h,1}; \dots; \tilde{q}_{t,h,G}] + b_h^Q\big)$ (8)
  for each stored chunk j: $\sigma_{t,h}(j) \leftarrow u_{h,t}^\top s_{h,j}$ (9)
  $S_{t,h} \leftarrow$ the k chunks with the highest $\sigma_{t,h}(j)$ (9)
  if the store is in host memory: copy the keys and values of the chunks in $S_{t,h}$, head h, to the GPU
  for each query head g of the group **do**
    $R \leftarrow P \cup \big(\bigcup_{j \in S_{t,h}} C_j\big) \cup W_t$ (5)
    $a_{t,h,g} \leftarrow$ attention (1) of $q_{t,h,g}$ over R, rotary encoding included; weight 0 outside R
  end for
end for

Algorithm B.2 Adding chunk j to the store, in one attention layer

when the keys and values of L further positions enter the store as chunk j:
for each key-value head h **do**
  keep $K_{h,j}$ and $V_{h,j}$ in GPU memory, or in host memory once the store outgrows the GPU Section 3.5
  $s_{h,j} \leftarrow W_h^K \psi(\tilde{K}_{h,j}) + b_h^K$, computed once and reused by every later query (7)
end for

B.1 Position-Free Scoring

Qwen3.5 applies rotary encoding to only part of each head's coordinates. Write a key as $k = (k^{\text{rot}}, \tilde{k})$, where $\tilde{k}$ holds the d_f position-free coordinates, and write $\mathcal{R}_t$ for the rotation that rotary encoding applies to the other coordinates at position t; YaRN rescales the frequencies of $\mathcal{R}_t$. The inner product of (1) between query position t and key position i then splits into a positional and a position-free term:

$$(\mathcal{R}_t\, q^{\text{rot}})^\top (\mathcal{R}_i\, k^{\text{rot}}) + \tilde{q}^\top \tilde{k} = q^{\text{rot}\top}\, \mathcal{R}_{i-t}\, k^{\text{rot}} + \tilde{q}^\top \tilde{k}. \tag{B.1}$$

By (7) to (9), the score $\sigma_{t,h}(j)$ is a function of $\tilde{K}_{h,j}$ and $\tilde{q}_{t,h,1}, \dots, \tilde{q}_{t,h,G}$ through ψ, W_h^K, b_h^K, W_h^Q and b_h^Q alone, none of which involves a position or $\mathcal{R}$. The selection $S_{t,h}$ therefore depends neither on where the query and the chunks lie nor on the YaRN scale factor s. What a chunk contains still decides where it ranks, since the model computes its keys from the whole input. A scorer on the rotated coordinates would instead depend on $i - t$ through $\mathcal{R}_{i-t}$, and at 4M tokens it would meet distances about four times the longest it saw in task training.

B.2 The Chunk Summary

For a position-free query $\tilde{q}$ and a chunk with position-free keys $\tilde{k}_1, \dots, \tilde{k}_L$, let $\bar{k}$ be their mean and $k^{\max}$ and $k^{\min}$ their coordinatewise maximum and minimum, the three blocks of ψ in (7). The summary holds both the logit of the query on the chunk's centroid and an upper bound on the largest logit inside the chunk:

$$\tilde{q}^\top \bar{k} = \frac{1}{L} \sum_{i=1}^{L} \tilde{q}^\top \tilde{k}_i, \qquad \max_{1 \le i \le L} \tilde{q}^\top \tilde{k}_i \le \sum_{c=1}^{d_f} \max\big(\tilde{q}_c\, k_c^{\max},\ \tilde{q}_c\, k_c^{\min}\big). \tag{B.2}$$

The bound holds term by term, since $k_c^{\min} \le \tilde{k}_{i,c} \le k_c^{\max}$ for every key i; Quest (Tang et al., 2024) computes the same bound from the same extrema. The two quantities part in the case that Section 3.3

describes. If one key scores $\tilde{q}^\top \tilde{k}_{i^*} = M$ and the other $L-1$ score about zero, the centroid logit is about M/L, diluted by the chunk length, while the bound stays at least M; a chunk of uniformly weak matches with the same mean key has the same centroid logit but different extrema. The router does not evaluate (B.2): (7) maps ψ linearly to $s_{h,j}$, and distillation learns how to weigh the three blocks. Whatever L is, a chunk enters scoring as r numbers per key-value head, against $2Ld$ for its keys and values.

B.3 Initialization

At initialization, the score of chunk j is the logit of the group's summed queries on the chunk's mean key, after a fixed random orthonormal projection Ω into the router's r-dimensional space:

$$\sigma_{t,h}(j) = \frac{1}{\sqrt{r}} \Big(\Omega \sum_{g=1}^{G} \tilde{q}_{t,h,g} \Big)^\top (\Omega \, \bar{k}_{h,j}). \tag{B.3}$$

In the terms of (7) and (8), $W_h^Q = [\Omega \cdots \Omega]$ with one block per query head of the group, $W_h^K = [\Omega \ 0 \ 0]$, which gives the extrema zero weight, and both biases are zero. Since $\Omega^\top \Omega$ is a projection, the expectation of (B.3) over Ω is proportional to $\sum_g \tilde{q}_{t,h,g}^\top \bar{k}_{h,j}$, the centroid estimate of the group's attention logits on the chunk. Distillation then trains every parameter of the router, the weights on the extrema included.

B.4 Cost

Table B.1 counts the work that one query at position t costs in one attention layer, in multiply–accumulates for products and in comparisons for the top-k; m_t denotes the number of chunks in the store when the query is read.

Table B.1: Work per query in one attention layer.

Step	GQA + YaRN	Link
Key–query products and value sums, (1)	$2Htd$	at most $2HBd$
Router: map of the group's queries, (8)	—	$Hd_f r$
Router: scores of the stored chunks, (9)	—	$H_k m_t r$
Router: top-k of the scores, (9)	—	$O(H_k m_t)$
Chunk summaries, (7), computed once per chunk	—	$3H_k d_f(1 + r/L)$ per position

Over a session of n tokens, full attention performs $Hd\, n(n{+}1)$ multiply–accumulates. Link's attention costs at most $2HBdn$, its query maps $Hd_f rn$ and its summaries $3H_k d_f(1 + r/L)\, n$, all linear in n. The one term that grows faster is the router's scoring, and since $m_t \le t/L$ it is a fixed fraction of full attention's work at every length:

$$\frac{\sum_{t=1}^{n} H_k m_t r}{\sum_{t=1}^{n} 2Htd} \le \frac{H_k r\, n(n+1)/(2L)}{Hd\, n(n+1)} = \frac{r}{2GLd}. \tag{B.4}$$

Attention's share of full attention's work, B/t for a query at position t, halves each time the input doubles, while the bound (B.4) stays where it is. Figure B.1 shows the measured counterpart. A timed read is every forward pass over the session, with the GPUs synchronized before and after; model loading, tokenization and answering are excluded, and Link's read includes the router and, from 2M on, every transfer between the host-memory store and the GPU.

From 1M to 2M and from 2M to 4M, Link's time per token grows by factors of 1.27 and 1.11 and that of GQA + YaRN by 1.84 and 1.91; each factor is $2^{\gamma-1}$ for the local exponent γ of (20). When its cache exceeds one GPU, GQA + YaRN splits its layers across GPUs that run in turn, so its read time is that of one GPU doing all of the work. Splitting the sequence instead could shorten the read but not reduce the total work, so the ratio of GPU time between the two reads is at least the ratio of their read times: 3.9, 5.7 and 9.9 at 1M, 2M and 4M.

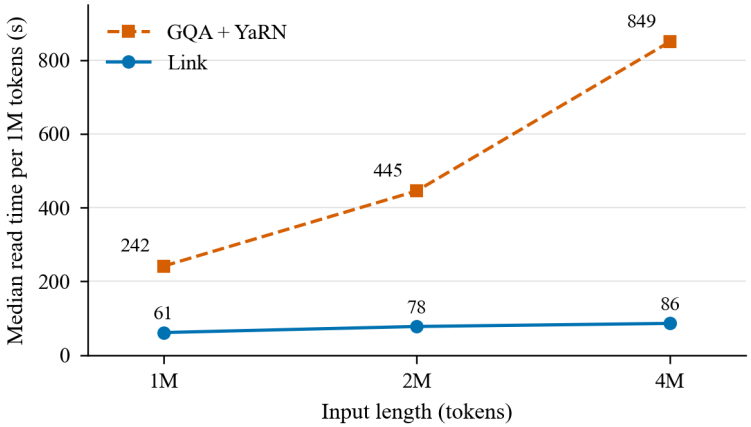


Figure B.1: Median read time divided by input length, for the seed-0 models of Table 9. A read whose time grows linearly with the input gives a flat line; one whose time grows with its square doubles with every doubling of the input.

B.5 Memory

Table B.2 compares, in stored numbers, the state that each system keeps in one attention layer.

Table B.2: State per attention layer.

State	GQA + YaRN	Link
Keys and values of the input	$2nH_kd$, on the GPUs, (3)	$2(\|P\| + \|W_t\|)H_kd$ for the exact window, on the GPU; $2mLH_kd$ for the store, on the GPU or in host memory
Keys and values that one query reads	all of the above	the exact window and $2kLH_kd$ from the store, on the GPU
Router summaries	—	mrH_k

With the store in host memory, the key-value state that a query needs on the GPU is $2(|P| + kL + |W_t|)H_kd = 2BH_kd$ per layer, whatever n is, which is (12). Two quantities grow with n: the store, $2mLH_kd$, and the router's summaries, mrH_k, a fraction $r/(2Ld)$ of the store.

Appendix C. Training

Figure C.1 sets out how the three compared systems are trained. The recorded configurations of a Link run and a GQA + YaRN run with the same seed differ in two entries only, the read and the number of GPUs, and every scored answer records the checksum of the weights it ran with; the checksums match the checkpoints that the validation rule chose.

C.1 The Router's Distillation

The target (13) is the group's causal attention restricted to the store and renormalized: chunk j's share of the attention that the G query heads place on stored chunks rather than on the exact window. The weight in (14) is proportional to the attention mass on the store, averaged over the group,

$$w_{t,h} \propto \mu_{t,h} = \frac{1}{G}\sum_{g=1}^{G}\sum_{j}\sum_{i\in C_j}\alpha_{t,i}^{(g)} \in [0,1], \tag{C.1}$$

so that a query attending mainly inside its exact window, which Link reads in full, weighs little. When the router's scores order the chunks as $p_{t,h}$ does, $S_{t,h}$ holds the k chunks with the largest target shares, the k-subset that captures the most of the model's attention on the store. The target is computed over

all coordinates, position-encoded ones included, while the scorer sees only the position-free ones. The distillation run used none of the benchmark's questions, and no answer label entered it or the choice of its checkpoint. Every Link model in the main comparison starts from its router, and so does the LongBench v2 system of Section 8.2.

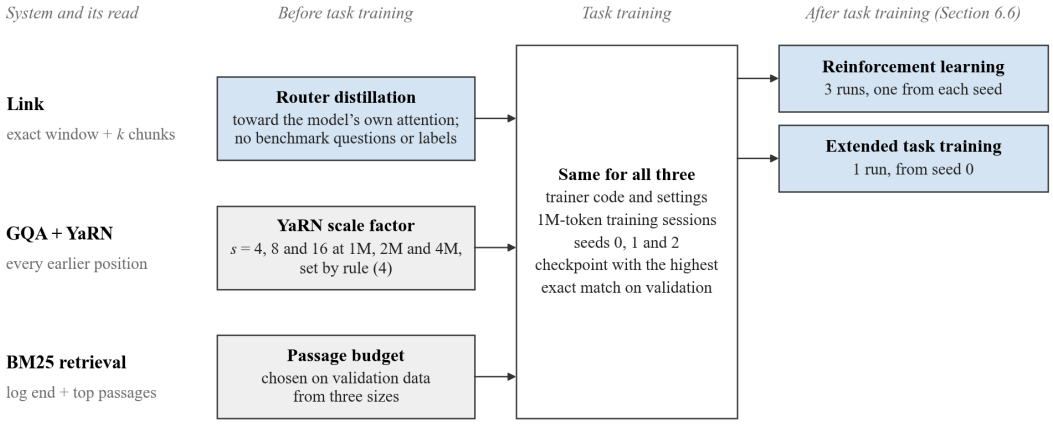


Figure C.1: Training of the three compared systems, all from Qwen3.5-9B. Blue marks steps that Link alone takes and grey the steps of one baseline. Link and GQA + YaRN train on the same sessions in the same order, step for step. Every checkpoint of task training and of the further training is chosen on validation data alone.

C.2 Reinforcement Learning

A rollout τ for a training question x consists of the read's selections S_t for its queries and the tokens y_i of the answer, so its log-probability (17) has one term for each of the two parameter groups. The estimator (19) is unbiased for the gradient of the expected reward at x: each rollout's baseline is computed from the other $K-1$ rollouts, which are independent of it, and $\mathbb{E}_\tau[\nabla \log \pi_{\phi,\theta}(\tau \mid x)] = 0$, so the baseline term vanishes in expectation and the likelihood-ratio term remains (Williams, 1992). Three further properties follow.

- It needs $\nabla \log \pi_\phi(S_t \mid q_t)$ only for sampled selections, which exists for a stochastic selection with differentiable probabilities; the deterministic top-k of (9) passes no gradient to unselected chunks, (10).
- A rollout's weight $R(\tau_a) - \beta_a$, where β_a is its leave-one-out baseline, is positive exactly when it earns more than the mean of its siblings, and a question whose K rollouts all earn the same reward contributes no update.
- Because correctness dominates the reward, any correct rollout a and incorrect rollout c of the same question satisfy $(R(\tau_a) - \beta_a) - (R(\tau_c) - \beta_c) = \frac{K}{K-1}\big(R(\tau_a) - R(\tau_c)\big) > 0$, so every correct rollout outweighs every incorrect one.

Validation and testing use the policy's greedy read, which selects as in (9). At the start of the stage it is the task-trained read, and before its first update each run checked, on a sample of validation questions, that the greedy read reproduces its starting model's answers token for token.

Appendix D. The Benchmark

Figure D.1 follows the recorded episodes to the questions that are scored. The leakage filters of Section 5.3 act on the test questions, and the stricter never-seen subset of Table 6 also requires the titled task named by a next-task question to be never-seen.

Figure D.2 shows the prompt, and Table D.1 the question types that occur. Of the nine combinations of how a question names its task and what it asks for, seven occur among the scored questions.

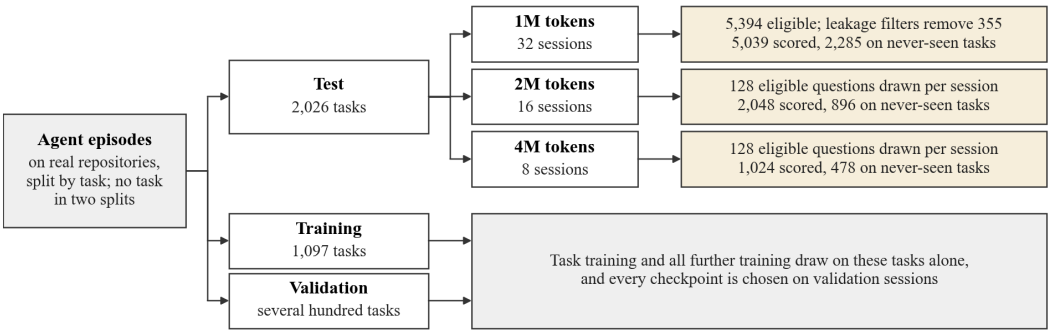


Figure D.1: From recorded agent episodes to scored questions. Never-seen tasks appear nowhere in the data used to train or validate Link, the router's run included.

```
<the log's one-line header>
<the log: one episode per task, in order>
<end-of-log marker>
<question block: one question about one earlier
task>
Reply with only the answer, on a single line.
Answer:
```

Figure D.2: Layout of the prompt that every compared system sees, in plain text with no chat template.

Table D.2 applies the scoring rule of Section 5.2 to replies for one gold answer.

Figure D.3 gives the distribution of answer distances. At every length the answers are spread over the whole log rather than gathered near its end.

Table D.3 lists the evaluation choices with the point at which each was fixed.

Table D.1: Question types. Rows give how a question names its task; columns give what it asks for.

Task named by	Path given when first opening a file	First exception line printed	First non-blank line of the first real code edit
Its exact title	asked	asked	asked
The first file opened in it	never asked	asked	asked
As the task immediately after a titled one	asked	asked	not among the scored questions

Table D.2: The rule applied to replies for the gold answer `repo/src/pkg/io.py`.

First line of the reply	Scored	Why
`repo/src/pkg/io.py`	correct	exact match
`src/pkg/io.py`	correct	shorter form that keeps `pkg/io.py`
`pkg/io.py`	correct	shorter form that keeps `pkg/io.py`
`work/repo/src/pkg/io.py`	correct	longer form
`` `src/pkg/io.py` ``	correct	backticks stripped, then a shorter form
`pkg/io.py.`	correct	trailing period stripped, then a shorter form
`io.py`	incorrect	drops the last directory

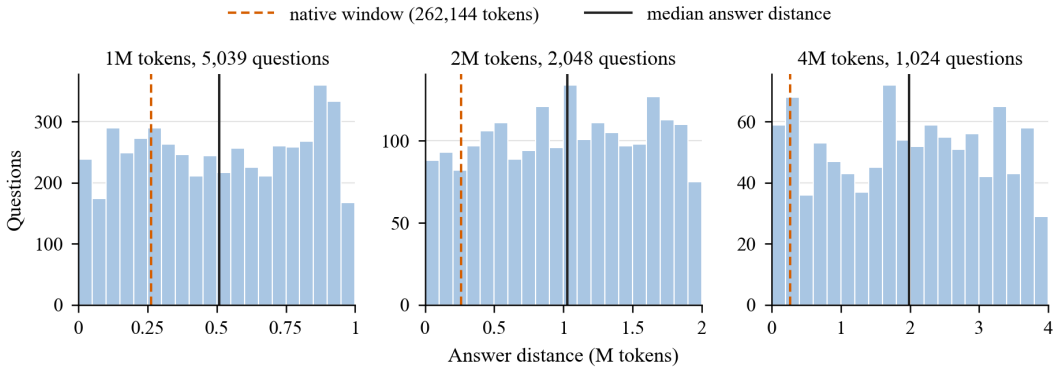


Figure D.3: Answer distances of the scored questions, in 20 equal bins per length. The dashed line marks Qwen3.5-9B's native window of 262,144 tokens, and the solid line the median: 508K, 1.03M and 1.98M tokens.

Table D.3: Evaluation choices and when each was fixed.

Choice	Fixed
Eligible questions, the scoring rule with its path clause, the leakage filters	before task training of the compared models
Task-training checkpoint rule: highest validation exact match, ties to the earlier	before task training of the compared models; recorded in each run's configuration
Size of the read, about 6,200 positions per query and key-value head	after exploratory development runs, some scored on the 1M test sessions; the same in every compared run and at every length
YaRN factors of 4, 8 and 16 at 1M, 2M and 4M	by (4) alone, before full attention was run at 2M or 4M
Rule for a lead to stand: all nine seed pairings, interval excluding zero	before each replication was trained
Draw of 128 questions per session at 2M and 4M	before any baseline result at those lengths
Never-seen subset	from data manifests alone, without reference to any answer
Budget of the BM25 baseline, 98,304 tokens of the log per question	on validation data, from three sizes, before any test run
Three GPUs for GQA + YaRN at 4M	for memory headroom, before any 4M run
Stopping of the reinforcement-learning runs	on validation data alone, before any run was tested
Checkpoint rule of the reinforcement-learning runs	in writing, before any run was tested
The three reinforcement-learning checkpoints	before any of their test answers was scored or read; tested once
Checkpoint of the extended training run	by the same validation rule, before testing
Router training	without any of the benchmark's questions or answer labels

Appendix E. Statistical Procedures

E.1 The Interval for a Lead

All intervals and tests are computed within one length, and the systems are paired by question and session. For Link runs $i = 1, 2, 3$ and baseline runs $j = 1, 2, 3$, let x_i and y_j be scores over the length's scored questions, $\bar{x}$ and $\bar{y}$ their means, and s_x^2 and s_y^2 their between-seed variances. The session bootstrap resamples the length's test sessions with replacement, recomputes the seed-averaged paired difference on the questions of the resampled sessions, and takes $\hat{\sigma}_b^2$ as its variance across resamples. Interval (15) adds this session variance to $(s_x^2 + s_y^2)/3$, the variance of $\bar{x} - \bar{y}$ that comes from the training runs:

$$(\bar{x} - \bar{y}) \pm t_{0.975,\nu} \sqrt{\frac{s_x^2 + s_y^2}{3} + \hat{\sigma}_b^2}, \qquad \nu = \frac{\left(s_x^2/3 + s_y^2/3\right)^2}{\frac{(s_x^2/3)^2}{2} + \frac{(s_y^2/3)^2}{2}}. \tag{15}$$

Each system's three runs are treated as an independent sample rather than paired by seed number, although same-seed runs share a data order (Section 5.5). With two degrees of freedom per system, ν lies between 2, when one system's between-seed variance dominates, and 4, when the two are equal. The multiplier $t_{0.975,\nu}$ therefore lies between $t_{0.975,4} = 2.78$ and $t_{0.975,2} = 4.30$, against 1.96 for a normal interval.

E.2 Other Intervals

The remaining intervals in the paper follow the procedures in Table E.1.

Table E.1: Intervals other than the leads of Table 5.

Quantity	Procedure
Leads on the subsets of Section 6.3, and the lead over BM25	(15), on the questions concerned
Lead of Link with reinforcement learning over GQA + YaRN	(15)
Gain of reinforcement learning over its starting models	session bootstrap alone, with the trained runs taken as given
Memorization control	bootstrap over the 32 test sessions at 1M
BABILong scores and differences	bootstrap over questions within each task and length

Appendix F. Results by Question Type

Figure F.1 breaks the comparison of Table 5 down by question type. Link scores above GQA + YaRN on 20 of the 21 combinations of type and length; the exception is the path of a next task at 1M, 59.9 against 60.5. Its lead is largest on edit-line questions, between +25.5 and +30.2 points at every length. At 4M the two types that name their task as the one after a titled task, and so add a second hop, are the hardest for both systems: Link answers 32.3% and 23.0% of them, GQA + YaRN 20.6% and 10.6%.

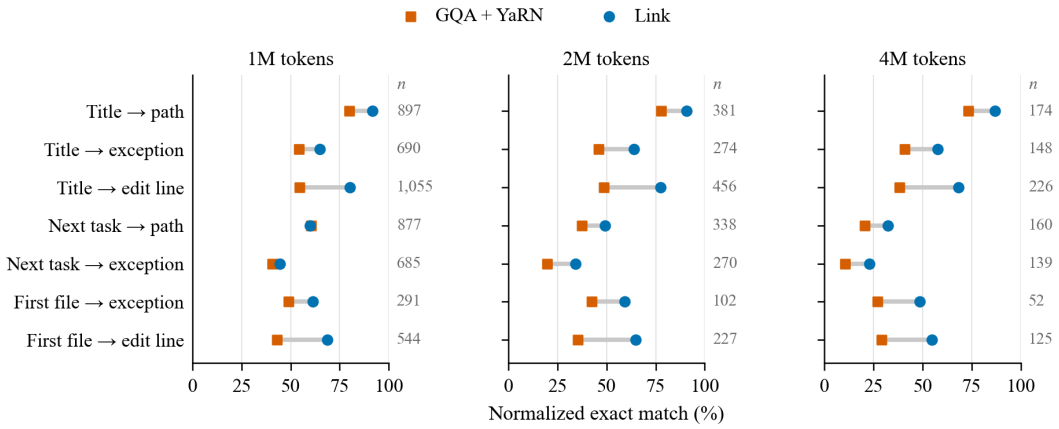


Figure F.1: Normalized exact match by question type, three-seed means after identical task training. Rows give how a question names its task and what it asks for (Table D.1); *n* is the number of scored questions of that type.

Appendix G. The Public-Benchmark Systems

Both systems combine Link's read with BM25, and both were scored on the official test sets with the official scorers (Section 8); Table G.1 compares them.

Table G.1: The two public-benchmark systems.

	BABILong	LongBench v2
Test set	qa1 to qa5 of the official test split, 100 questions per task and length	all 503 questions
Prompt and scoring	the benchmark's answer-matching scorer	the benchmark's zero-shot chain-of-thought prompt, answer extraction and scorer
Model	Qwen3.5-9B with Link, trained on the task	Qwen3.5-9B with Link in thinking mode, using the router from which every Link model of the coding-log comparison starts
Evidence beside Link's	passages found by keyword search	passages found by keyword search
Answer	direct; on qa2, qa3 and qa5, a short fact trace whose final sentence is scored; on qa3 at the longer inputs, a vote over five samples	self-consistency over up to nine sampled reasoning chains
Configuration chosen on	validation stories	793 questions from QuALITY and DetectiveQA

G.1 BABILong

Section 8.1 reports the system at 512K and 1M tokens, beyond the model's native window. On inputs up to 128K, which fit inside that window, we report its final version, which writes short fact traces on qa2, qa3 and qa5; we score their final sentence. Figure G.1 gives the accuracy at every length, and Figure G.2 that of every task at every length, each beside untrained Qwen3.5-9B answering from keyword-retrieved excerpts.

As the benchmark's leaderboard does, we average the nine lengths from 0K to 128K. The final version answers 96.2% of these questions (4,329 of 4,500; 95% CI [95.7, 96.7]), against 51.0% for the untrained reference: between 96.2 and 98.4 at every length up to 32K, then 94.2 at 64K and 90.6 at 128K, and 97.1, 96.0, 89.1, 99.3 and 99.4 on qa1 to qa5.

We report this figure as a development result. The system was developed over several rounds, each scored on the test split and informing the next. Every setting of the final configuration, including a checkpoint and an answer procedure for each task and length range (for qa3 on the longer inputs, a vote over five sampled answers), was chosen on validation stories before that configuration was scored on test. Its task training includes the benchmark's training split, which contains most of the qa4 test stories; without qa4, the 0K to 128K average is 95.4.

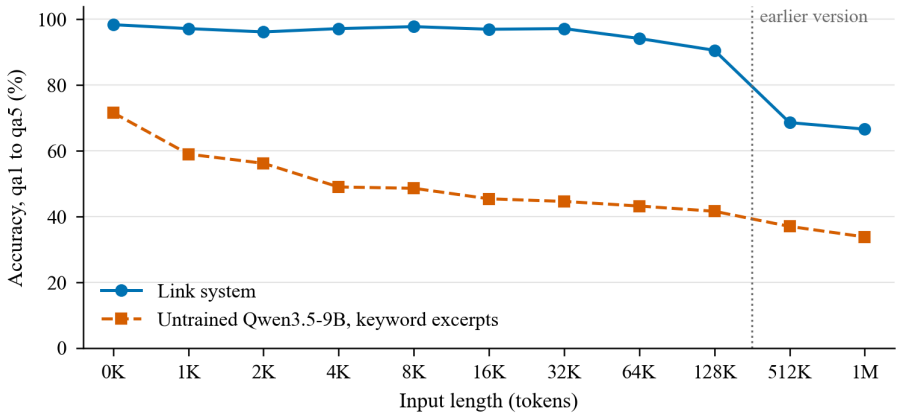


Figure G.1: BABILong accuracy by input length, averaged over qa1 to qa5 (500 questions per point, 100 per task), on the official test split with the official scorer. The reference line is untrained Qwen3.5-9B answering from keyword-retrieved excerpts of the same inputs, at one excerpt size throughout, as in Table 10. Points to the right of the dotted line come from the earlier version, the one reported in Table 10.

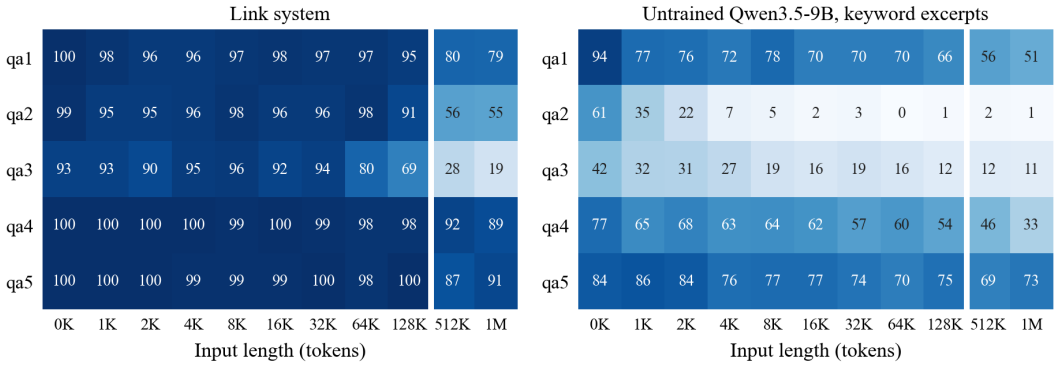


Figure G.2: BABILong accuracy (%) by task and input length, 100 questions per cell, on the official test split with the official scorer. Left, the Link system: its final version from 0K to 128K and, after the gap, the earlier version without fact traces at 512K and 1M. Right, untrained Qwen3.5-9B answering from keyword-retrieved excerpts, at the one excerpt size of Figure G.1.

Up to 128K the final system scores at least 90% on every task and length except qa3, which needs three supporting facts, at 64K and 128K (80% and 69%). The untrained reference loses most on the tasks that need more than one fact: from 8K on it answers at most 5% of qa2 and at most 19% of qa3 questions. At 512K and 1M the earlier Link version answers 55–56% of qa2 and 19–28% of qa3.

At 512K and 1M, Table 10 compares the earlier version with Qwen3.5-9B trained on the same BABILong training questions for the same number of steps, with an adapter of the same size, each warm-started from its own earlier adapter. Both chose a checkpoint and a way of reading on validation questions alone: the control once for all tasks, answering from keyword-retrieved excerpts, and Link per task, in its second development round. Link's first round, with one setting for all tasks, led by +3.2 and +1.6 points, with intervals that include zero. Table G.2 gives the comparison by task. Link's lead comes mainly from qa2, whose answers need two supporting facts, and from qa3, which needs three; on the single-fact qa1 at 1M, the trained control scores higher.

Table G.2: BABILong at 512K and 1M by task (%), 100 questions per cell, for the earlier Link version and Qwen3.5-9B trained the same way.

Task	512K, Link	512K, trained Qwen	1M, Link	1M, trained Qwen
qa1	80	81	79	91
qa2	56	16	55	19
qa3	28	16	19	16
qa4	92	96	89	84
qa5	87	94	91	92
Average	68.6	60.6	66.6	60.4

G.2 LongBench v2

Link first reads the whole input and ranks its passages for the question. If the input fits the native window with room left for reasoning, the model then reads all of it, followed by passages chosen by Link and by keyword search. For longer inputs it reads the start and end of the input with Link's passages and fills the rest of its window with keyword-retrieved passages. The answer is decoded with self-consistency over up to nine sampled reasoning chains (Wang et al., 2023), and the system answers 308 of the 503 questions correctly.

The test questions had been used before: an earlier version of the system scored 58.3% on them, and other variants of Link were scored on them during development. The configuration reported here was chosen afterwards on 793 questions from QuALITY (Pang et al., 2022) and DetectiveQA (Xu et al., 2024), neither of which shares documents with LongBench v2, with each choice made under a rule fixed before the validation scores it relied on were used, and it was scored on the test questions once.