BROWSA: SEMANTIC AWARE SPARSE ATTENTION FOR WEB AGENTS WITH KV CACHE PACKING

Anonymous authors
Paper under double-blind review

ABSTRACT

Browser agents re-read tens of thousands of tokens of serialized DOM at every step, yet only a small fraction of that context influences the next action. Sparse attention can skip the rest, but existing methods drag in irrelevant neighbors and/or put substantial pressure on the hardware. We observe that web pages, system prompts, and action histories already carry semantic boundaries through HTML and other structured formats. BrowSA exploits these boundaries twice. It selects among chunks defined by the prompt's semantic boundaries and ranks them with a score calibrated for unequal chunk lengths. A placement stage then packs these chunks into physical KV pages, so the units the scorer selects are the units the kernel fetches. BrowSA reduces decode attention latency by up to 68% over dense attention while matching or exceeding its task success on WebVoyager, GAIA, and Online-Mind2Web.

1 INTRODUCTION

Browser agents turn a natural-language request into clicks, keystrokes, and navigation on live websites (He et al., 2024). A single request (e.g., book this flight, sort these results by price) unfolds over tens of steps, and at every step the model parses the page's Document Object Model (DOM), the agent's system prompt, and earlier interactions in the task (Zhou et al., 2024).

Unfortunately, today's browser agents are painfully slow. Modern web pages are large: even simple pages, once distilled into a DOM and stripped of JavaScript and CSS, routinely run to thousands or tens of thousands of tokens. Models must attend over all of this context to generate some reasoning and the next action (e.g., selecting from a dropdown menu or clicking a submit button), and they must do it again after every page transition. Decode cost scales with the size of the cached context, so a single request pays for the entire page tens of times over. The end result is that browsing tasks a human could finish in seconds frequently take agents orders of magnitude longer to complete.

What makes this wasteful is how little of that context matters. On one recorded Amazon search page, the 17 product titles the agent needs are 4.8% of the characters in a 16,415-token serialization; the rest is nested layout containers, tracking attributes, and per-card boilerplate. Worse, because serialized HTML is a depth-first traversal, even that 4.8% is scattered across thousands of short spans.

Sparse attention is a natural fit: during decode, each query reads only the part of the KV cache that matters, and because decode attention is memory bound, every skipped token cuts latency. Existing methods, however, force a tradeoff of selection granularity against page locality (Section 2.1).

We observe that web pages already carry natural semantic boundaries in HTML's heavily structured format. Figure 2 illustrates this with a recorded WebVoyager task (He et al., 2024) that requires sorting results by Best Sellers. Arbitrary, fixed KV page boundaries can split the one option that matters across two pages, forcing the model to load both to attend to it. In contrast, the option's enclosing HTML element provides a more natural boundary. Moreover, this structure extends beyond the DOM: browser agent system prompts are organized into sections, and action histories are hierarchical and frequently serialized as JSON.

Our system, BrowSA, exploits this structure in two ways. At selection time, it partitions the prompt along semantic boundaries rather than fixed token offsets. At placement time, it reuses those boundaries to pack chunks into physical KV pages, so the units the scorer selects are the units the kernel fetches. Semantic units raise two problems that fixed pages avoid (Section 2.2): (1) their lengths vary by orders of magnitude, which inflates extrema-based scores for long chunks

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

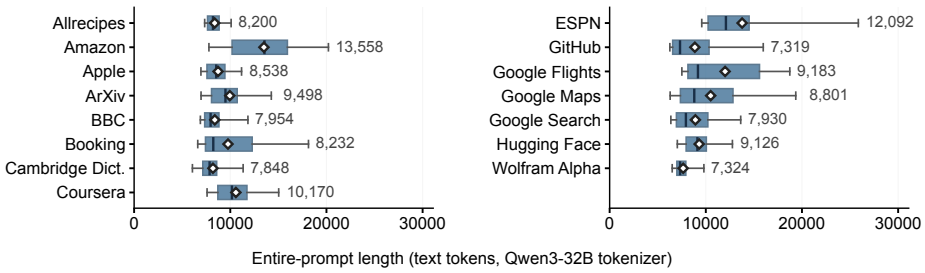

Figure 1: **Browser agents attend over long prompts at every step.** Per-step prompt length (system prompt, history, agent state, and DOM) for each WebVoyager website, over 5,955 agent steps from dense runs of all 643 tasks. Median steps carry 7,319 to 13,558 tokens depending on the website, and the model re-reads this context to emit a few action tokens at each of a task's tens of steps. Boxes span the interquartile range with the median marked (annotated); whiskers span the 5th to 95th percentiles; diamonds mark the mean. Counts are text tokens under the Qwen3-32B tokenizer and exclude screenshot vision tokens.

(Figure 5), and (2) their boundaries rarely coincide with page boundaries. BrowSA answers the first with a length-calibrated ranking score (Section 3.1) and the second with query-independent packing (Section 3.2), so selection stays semantic while memory access stays page-aligned.

This paper makes three contributions: a semantic partition of the browser-agent prompt; a ranking score calibrated for chunks of unequal length; and a boundary-preserving KV placement stage that packs chunks into physical pages. We ground the chunking and scoring choices in recorded WebVoyager traces and controlled action replays, which isolate how selection alters the agent's next action. BrowSA reduces decode attention latency by up to 68% over dense attention while matching or exceeding its task success on WebVoyager, GAIA, and Online-Mind2Web.

2 Motivation

We first describe the costs of a browser agent step and review relevant sparse attention approaches before examining why selection units should follow the structure of the browser-agent prompt.

2.1 Browser agent latency and attention sparsification

Across 117 steps from 15 WebVoyager tasks (dense Qwen3-32B on a single B200, one request at a time), decoding accounts for 6.3 seconds of a mean 11.3-second step (56%, Figure 3b). Sparse decoding cuts the KV reads that dominate this time; existing designs differ not only in how they select but, critically, in the granularity at which they select.

For example, systems like Quest score fixed-size pages using an upper bound computed from per-channel key extrema (Tang et al., 2024), matching the KV cache layout but frequently promoting irrelevant tokens that happen to be on the same page as relevant ones. Systems like DeepSeek DSA (DeepSeek-AI, 2025) and Tactic (Zhu et al., 2026) take a different approach, allowing selection of non-contiguous tokens, achieving higher precision, but at the cost of inefficient memory access patterns. Tactic, for instance, clusters each head's keys during prefill and ranks clusters against the query at decode time; the clustering nonetheless costs time quadratic in context length and must be redone at nearly every step in a browser agent workflow, and it still leaves selected tokens scattered across physical pages. Modern kernels can sometimes hide this latency, but it still incurs high HBM traffic, increases TLB pressure, and interacts especially poorly with page-granular machinery outside the kernel, such as offloading, prefix sharing, and KV transfer in disaggregated serving.

2.2 On semantic boundaries

A good selection unit keeps an action target's identifier, label, and local context together (Figure 2). Figure 3a shows the complete "Sports" element from a recorded WebVoyager DOM: `[1790]<div role=listitem level=1 />` and its child text, 20 tokens in total. A fixed 16-token boundary splits the identifier and attributes from the tag ending and child text, and highly ranked attention positions fall on both sides of it. Keeping either block loses part of the element; keeping both adds 12 unrelated tokens. Appendix A.7 dives deeper into results and details.

We argue that the right unit is already defined by the content itself. On the page, it is a DOM element; in the system prompt and history, it is a section or paragraph of prose, or an object in structured JSON.

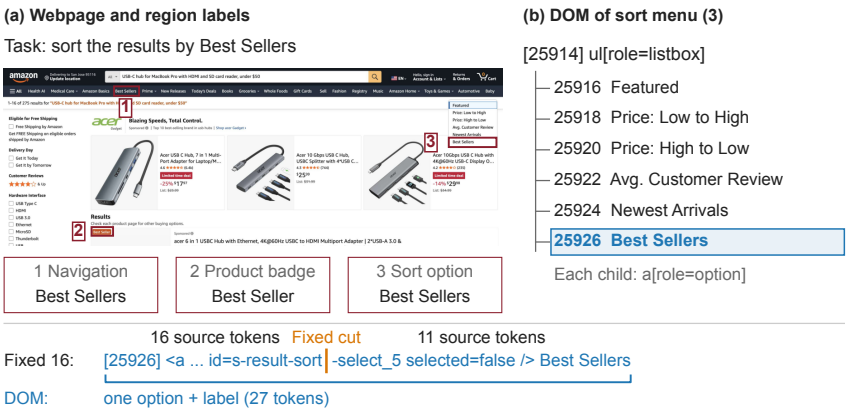

Figure 2: **A motivating example of DOM-based semantic chunking.** WebVoyager Amazon–27, step 3 (He et al., 2024). Labels below the screenshot transcribe (1) the *Best Sellers* navigation link, (2) a *Best Seller* product badge, and (3) the *Best Sellers* sorting option required by the task. DOM grouping keeps click target 25926 and its label together, while a local 16-token grid splits the option. Counts use the full source text, abbreviated here.

These boundaries come for free, since the serializer that produces the prompt already emits them. Adopting them, however, raises two issues that fixed pages avoid.

The first is variable chunk length, which in browser contexts spans orders of magnitude. Consider Quest's extrema-based ranking. The longer a chunk, the greater the chance that some token pushes each channel's extremum further out. Long, irrelevant chunks can then outrank short, decisive ones. The other common approach, scoring a chunk by its mean key or centroid (Zhu et al., 2026; Hooper et al., 2025; Lu et al., 2025; Sun et al., 2025), avoids this inflation but errs in the opposite direction: a single strong match, such as the label of the target option, is diluted by the tokens around it, increasingly so as chunks grow.

The second is the mismatch of semantic boundaries and KV cache pages. A relevant chunk can straddle pages shared with irrelevant neighbors, so a smaller logical selection does not always guarantee fewer page reads.

3 Design

BrowSA has two stages (Figure 4): structural chunking with calibrated ranking (Section 3.1) and query-independent KV placement (Section 3.2). Chunking and placement run once per generation call, while ranking runs at each decoding layer and step. Before prefill, BrowSA fixes one logical partition of the prompt. After prefill, it builds a physical layout for each layer and KV head from the cached keys. Both stay fixed for the call, while each decoding step is free to select different chunks.

3.1 Structural chunking

BrowSA partitions the serialized prompt into contiguous token spans that follow its structure. A lightweight parser builds a tree from signals that the serializer already emits, namely message boundaries, element identifiers, and indentation, and treats the remaining text as paragraphs. It also identifies the live DOM region in the agent observation, separating actionable page content from DOM examples inside instructions.

Chunks are extracted top-down under a size cap. A subtree that fits becomes one chunk. A larger subtree is split by descending into its children, and the parent's leading and trailing tokens are attached to the first and last resulting chunks so that no tokens are dropped. Small adjacent chunks under the same parent merge, and an oversized leaf can be split to respect the cap. The partition therefore favors structural boundaries without requiring every element to remain indivisible.

Figure 4a illustrates these rules with a 64-token chunk cap and a six-token merge target. The 36 token field subtree joins four opening form tokens to make C_1, while the 20 token button remains C_2. Two adjacent four token hints merge into C_3. The 100 token help leaf splits into 64 and 36 token spans, with four closing form tokens attached to the latter. The resulting five chunks cover all 172 tokens.

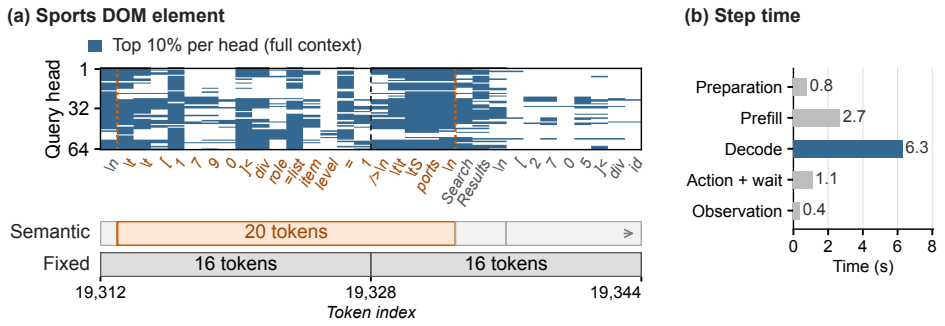

Figure 3: (a) A complete 20-token DOM element crosses two fixed 16-token blocks. Orange marks this chunk, while blue marks each query head's top 10% of attention positions over the full context. (b) Measured mean step times for dense serving, including all decoder work.

Length calibrated chunk ranking. BrowSA introduces a scalable ranking algorithm based on the observation that Quest's extrema envelope (Tang et al., 2024) splits exactly into two terms with different behavior: a signed center that measures how well the chunk aligns with the query, and a nonnegative width that measures how far its keys spread. The width grows on nested chunks, motivating a correction that leaves the signed center unchanged. Concretely, for chunk C_j with L_j keys, let $M_{j,h}$ and $m_{j,h}$ be the coordinate-wise key maxima and minima in KV head h, and let $\bar{q}_h$ be the pooled query for that head (defined below). Omitting the attention scale, the envelope decomposes as:

$$U_{j,h} = \sum_d \max(\bar{q}_{h,d} M_{j,h,d}, \bar{q}_{h,d} m_{j,h,d}) = \underbrace{\bar{q}_h^\top \frac{M_{j,h} + m_{j,h}}{2}}_{c_{j,h}} + \underbrace{\sum_d |\bar{q}_{h,d}| \frac{M_{j,h,d} - m_{j,h,d}}{2}}_{w_{j,h}}. \quad (1)$$

$U_{j,h}$ upper-bounds the chunk's true maximum $T_{j,h} = \max_{t \in C_j} \bar{q}_h^\top k_{t,h}$, with additive gap $G_{j,h} = U_{j,h} - T_{j,h} \geq 0$. Extrema from different tokens combine into a 'corner' of the region, so the gap can widen with length.

Figure 5a measures this inflation on contiguous windows of 8 to 256 tokens from five held-out tasks, isolating window length from the parser's boundary choices. The mean gap grows from 8.27 to 19.62 scaled logits (2.37×). Because the mean center is near zero, U closely tracks the width w. The trend is consistent with a conditional result: for i.i.d. Gaussian keys and an independent query, a nonzero expected gap grows asymptotically as $\sqrt{\ln L}$ (DasGupta et al., 2014) (Appendix A.1). We do not claim that this law holds for trained keys. Appendix A.2 details sampling and aggregation.

We therefore calibrate only the width and score each chunk separately in every KV head:

$$s_{j,h} = c_{j,h} + \left(\frac{A}{L_j}\right)^\alpha w_{j,h}. \quad (2)$$

The reference length $A > 0$ sets where the score equals the envelope, and $\alpha \geq 0$ sets the strength of the correction. Leaving the signed center untouched matters. Scaling the whole envelope would also pull negative centers toward zero, rewarding long chunks that point away from the query. Because extrema are cached after prefill, calibration adds only one scalar multiply per chunk at decode time. We fit A and α empirically rather than deriving them from the Gaussian model, and for $L_j > A$ the score is a ranking heuristic without an upper-bound guarantee.

With $A = 1$ and α fit on eight tasks, the mean absolute residual of task means on the remaining five falls from 14.11 to 1.55 scaled logits across all six tested lengths (Figure 5b). Long windows become underestimated on average. This diagnostic measures score error only, not selection quality or task success. Appendix A.2 details the protocol.

Admission. At each layer and decoding step, each KV head h ranks prompt chunks by $s_{j,h}$ and admits the highest-ranked chunks within the selection ratio, with selected indices denoted by $\mathcal{S}_h$. Let G_t contain the protected prefix, recent tokens, and all tokens generated so far in the current call, including the token processed at decoding step t. The retained positions are $\mathcal{T}_h = (\bigcup_{j \in \mathcal{S}_h} C_j) \cup G_t$, and we report the resulting token and page counts separately. The choice of whether to reserve these tokens for attention computation is orthogonal to BrowSA. For the example query $q = 1$ in Figure 4a, the illustrated selection takes the top two chunks, C_5 and C_3, containing 48 tokens in total.

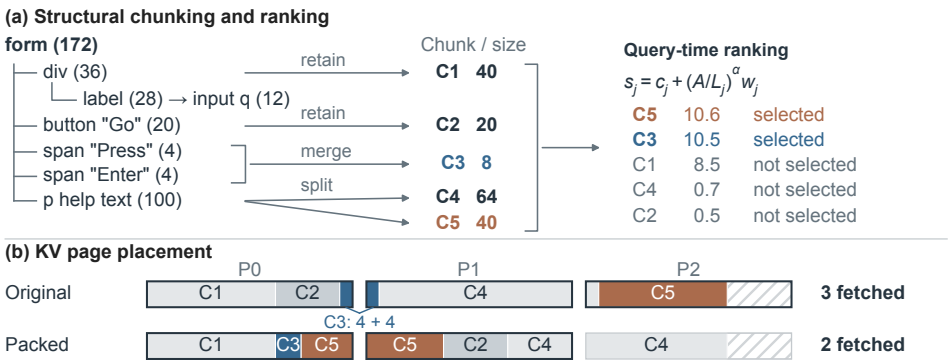

Figure 4: **BrowSA overview on a constructed example.** (a) Structural chunking with calibrated query-time ranking. (b) Fixed placement reduces fetched pages for the same selection, excluding protected and generated tokens. Hatching marks unused capacity. Appendix A.6 specifies the toy keys and parameter settings.

Mean query pooling. For N chunks, scoring every query head costs $O(NH_{\mathrm{Q}}d)$. Under grouped-query attention, BrowSA instead averages the query heads $\mathcal{G}_h$ that share KV head h,

$$\bar{q}_h = \frac{1}{|\mathcal{G}_h|} \sum_{a \in \mathcal{G}_h} q_a, \tag{3}$$

which reduces scoring to $O(NH_{\mathrm{KV}}d)$. Because the width term uses $|\bar{q}_h|$, scoring the pooled query is not the same as averaging per-head scores, and it can smooth over head-specific preferences. Pooling affects selection only. Attention still uses the original queries $Q = \{q_a\}_{a=1}^{H_{\mathrm{Q}}}$.

3.2 Query-independent KV placement

Ranking chooses the right chunks; placement makes them cheap to fetch. Following PagedAttention (Kwon et al., 2023) and LServe (Yang et al., 2025d), BrowSA decouples logical selection from physical pages. It then permutes the cache so that chunks likely to be selected together share pages. Algorithm 1 (Appendix A.4) lists the complete execution order.

Worst-case score distance. Dropping layer and head indices, Equation 2 scores chunk C_j as:

$$b_j = \frac{M_j + m_j}{2}, \qquad u_j = \left(\frac{A}{L_j}\right)^{\alpha} \frac{M_j - m_j}{2}, \qquad S(q, C_j) = q^{\top} b_j + |q|^{\top} u_j. \tag{4}$$

For $x = b_a - b_b$ and $y = u_a - u_b$, define

$$D_{\mathrm{worst}}(C_a, C_b) = \max\left\{ \left\| [|x| + y]_+ \right\|_2, \left\| [|x| - y]_+ \right\|_2 \right\}, \tag{5}$$

where absolute values and $[\cdot]_+$ act coordinate-wise. This is exactly the largest absolute calibrated score gap per unit query norm (Appendix A.5). Small distance indicates similar scores across query directions without observing a decoding query. It does not guarantee joint selection.

Placement and retrieval. BrowSA clusters chunks by average linkage over these distances (Müllner, 2011). Every movable chunk starts as its own group, and each step merges the pair of groups with the smallest mean cross-group distance. Distances to the remaining groups are updated as a chunk-count-weighted average of the merged pair's distances. At each node, the child containing the smaller original chunk index goes first. The leaves are then concatenated without padding, so merged groups stay contiguous but may cross page boundaries. Configured boundary chunks at the start and end of the prompt keep their offsets, so only the middle is reordered. The layout stays fixed as the recent-token window advances. Keys and values receive the same permutation, which preserves within-chunk order and original positional encodings. The map $\pi_{\ell,h}(i)$ gives the physical page of logical token i in layer ℓ and KV head h, with P slots per page.

In Figure 4b, the order C_1, C_3, C_5, C_2, C_4 places the selected C_3 and C_5 together, reducing their footprint from three pages to two. Both layouts retain all 172 tokens in three 64-slot pages. This unrestricted example omits protection and generation. Packing targets fetched pages rather than allocated capacity.

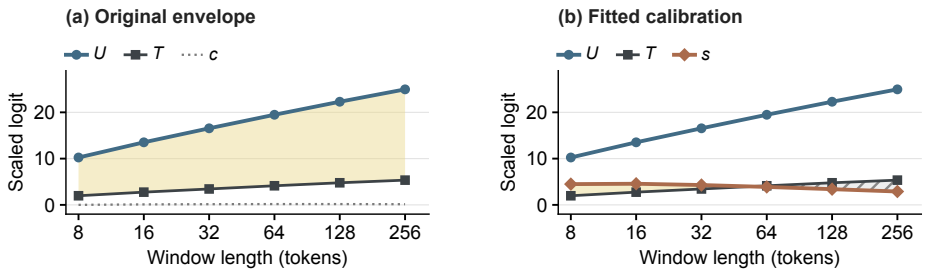

Figure 5: **Score calibration on five held-out tasks.** (a) Original U, T, c. (b) Fitted s with the same U, T. Yellow and hatching mark overestimation and underestimation of the mean curves. All six lengths are shown. Fitting protocol: Appendix A.2.

During decoding, each layer appends the new token's keys and values and then scores chunks. It maps the selected positions and G_t through $\pi_{\ell,h}$ to a deduplicated set of pages per KV head. Attention uses each head's original query over all valid entries on the fetched pages. The backend masks empty slots and derives causal visibility from logical positions, so relocating an entry never changes its positional interpretation. To maximize GPU memory efficiency, unselected neighbors on a fetched page are attended as well. Packing can therefore change which extra tokens are attended alongside the selection, but it never drops a selected token. For n prompt tokens, each layer and head retains all $\lceil n/P \rceil$ prompt pages, and only the number fetched depends on the selection. Generated entries extend the page maps without altering prompt chunks or their summaries.

Setup cost. For N movable chunks per layer and KV head, pair distances cost $O(N^2 d)$ and average linkage uses $O(N^2)$ time and memory (Müllner, 2011). This cost is quadratic in chunks rather than tokens and needs no iterative refinement, but it still makes chunk count matter for setup as well as for per-step scoring. A GPU builder keeps the condensed distance matrix, nearest-neighbor-chain linkage, and permutations on device. The serving backend then permutes keys and values together and installs head-specific page metadata. Orientation and offsets take $O(N)$ work, and KV copying scales with the number of moved tokens. Setup and KV copies are paid once and can be amortized over decoding. Appendix A.4 details construction and the serving variants.

Prefix caching. Finally, we note the interaction of BrowSA with prefix caching mechanisms, which are only able to reuse pages if the shared prefix is consistent across invocations (Kwon et al., 2023; Zheng et al., 2024). When chunks are clustered over the whole prompt Zhu et al. (2026), new tokens at the end can change the pairwise distances and thus which chunks share a page, so even the pages holding an unchanged system prompt stop matching the cache.

BrowSA can preserve reuse by packing each section of the prompt independently (e.g., the system prompt versus the DOM) and relying on two properties: (1) the fact that BrowSA's packing strategy is dependent only on the tokens being packed and not the query, and (2) the fact that each section of the prompt, including the DOM, arrives whole on each step. By (1), the pages, chunk summaries, and page maps of a section are then determined by the prompt up to the section's end, just as an ordinary page is determined by the tokens up to its own end. By (2), a section is only ever packed once—later steps either repeat it verbatim or replace it entirely. The cache therefore matches whole sections rather than individual pages, since an early page of a packed section can hold tokens from its end, but can otherwise reuse the prior results without repacking. We disable prefix caching in all experiments to isolate the effect of packing and evaluate the base scheme of Section 3.2.

4 Experiments

Experimental setup. We evaluate interactive task success on three live-web benchmarks: WebVoyager (He et al., 2024) (643 tasks across 15 websites), GAIA (the 90-task web subset used by WebVoyager) (Mialon et al., 2024; He et al., 2024), and Online-Mind2Web (Xue et al., 2025) (300 tasks across 136 websites). We use Qwen3-32B (Yang et al., 2025a) on a single NVIDIA B200 unless otherwise specified. We compare BrowSA with attention over the full KV cache (Dense), Quest (Tang et al., 2024), and Block Sparse. Both sparse baselines use Vortex implementations (Chen et al., 2026b), which support batching and grouped query attention (GQA). Following Mind2Web

			Page Size 32			Page Size 64		
Website	Tasks	Dense	BrowSA	Quest	Block Sparse	BrowSA	Quest	Block Sparse
Overall	643	46.2	**49.1**	45.9	44.9	**50.7**	40.7	45.7
Allrecipes	45	73.3	73.3	73.3	**80.0**	71.1	64.4	**77.8**
Amazon	41	61.0	61.0	**63.4**	53.7	61.0	51.2	**63.4**
Apple	43	53.5	**53.5**	44.2	46.5	**60.5**	48.8	41.9
ArXiv	43	46.5	**51.2**	48.8	44.2	**48.8**	34.9	**48.8**
BBC	42	54.8	57.1	54.8	**59.5**	**57.1**	50.0	54.8
Booking	44	11.4	**11.4**	6.8	9.1	6.8	**9.1**	**9.1**
Cambridge Dict.	43	76.7	67.4	**69.8**	58.1	**79.1**	55.8	72.1
Coursera	42	57.1	**71.4**	**71.4**	64.3	**69.0**	59.5	64.3
ESPN	44	34.1	**40.9**	27.3	31.8	36.4	29.5	**38.6**
GitHub	41	36.6	**51.2**	36.6	39.0	**65.9**	34.1	34.1
Google Flights	42	7.1	2.4	**4.8**	2.4	**2.4**	**2.4**	**2.4**
Google Maps	41	48.8	**41.5**	**41.5**	39.0	**39.0**	29.3	**39.0**
Google Search	43	72.1	76.7	74.4	**81.4**	72.1	72.1	**74.4**
Hugging Face	43	41.9	**46.5**	41.9	34.9	**51.2**	41.9	32.6
Wolfram Alpha	46	19.6	**32.6**	30.4	30.4	**41.3**	28.3	32.6

Table 1: **WebVoyager task success by website.** Success rate (%, higher is better) across 643 tasks at a 12.5% selection ratio. Bold marks the best sparse method in each size group, including ties.

2 (Gou et al., 2025) and Agent S3 (Gonzalez-Pumariega et al., 2026), we report Pass@3: a task counts as solved if any of three independent runs succeeds. For BrowSA, page size 64 uses chunk targets of 6 to 32 tokens, while page size 32 uses targets of 6 to 16 tokens. We control sparsity with a single *selection ratio*, the target fraction of the KV cache a sparse method attends at each decoding step.

4.1 Task Success across Selection Ratios

We first ask whether sparse attention preserves task success as the selection ratio shrinks and at different selection granularities. Across three benchmarks, BrowSA matches or exceeds Dense at every tested selection ratio, whereas Quest and Block Sparse fall behind Dense once the ratio is small.

WebVoyager. At a 12.5% selection ratio, BrowSA achieves the highest overall success among sparse methods with both page sizes (Table 1). Increasing page size from 32 to 64 tokens lowers Quest's success by 5.2 percentage points, consistent with larger blocks admitting more irrelevant neighbors (Section 2). BrowSA instead gains 1.6 percentage points. Its semantic selection units remain separate from physical pages, allowing larger pages without requiring selection of entire fixed blocks.

GAIA and Online-Mind2Web. Figure 6 compares selection ratios of 6.25% and 12.5% with page size 32. On GAIA, the sparse baselines roughly match Dense at the larger ratio but fall below it when the ratio is halved. BrowSA achieves 44.4% success at the smaller ratio, compared with 37.8% for Dense. Online-Mind2Web shows the same pattern, with BrowSA remaining above Dense while both sparse baselines fall below it. Halving the selection ratio costs BrowSA at most 1.2 percentage points on either benchmark, suggesting that its selected context retains the information needed for the next action.

Why sparse attention can beat Dense. The results suggest that removing less relevant context can sometimes improve the agent's decisions. One possible explanation is that sparsification suppresses distracting contributions from long page contexts, consistent with observations by Joshi et al. (2026a).

4.2 System performance analysis

To compare methods on identical inputs, all experiments in this section replay prompts from 1,283 recorded WebVoyager requests spanning all 643 tasks and serve Qwen3-32B on one B200.

Token-matched decoding latency. Table 2 compares methods at the same attended-token counts and similar selection granularity: Quest and Block Sparse use block size 32, and BrowSA selects

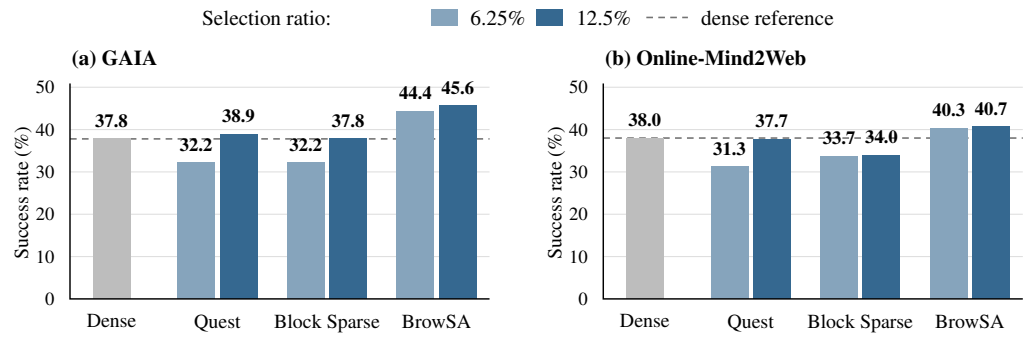

Figure 6: **Task success at two selection ratios.** Task success on (a) GAIA and (b) Online-Mind2Web at selection ratios of 6.25% (light) and 12.5% (dark). All methods use page size 32. Dense attends the full KV cache.

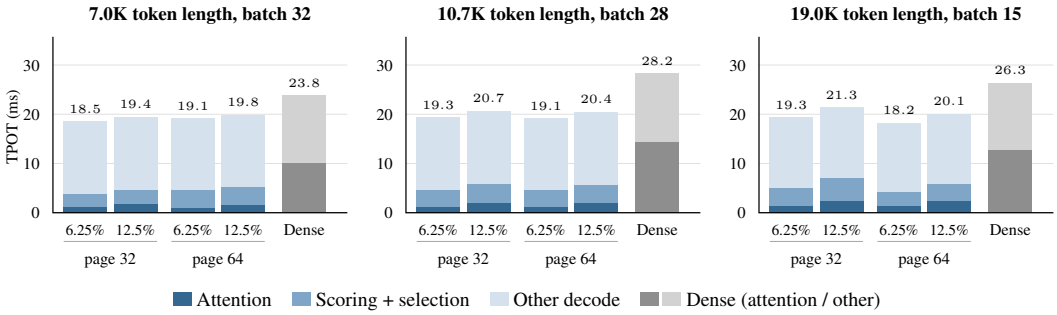

Figure 7: **Decode latency (TPOT, ms per output token) on recorded WebVoyager requests, by prompt length.** Panel titles give the median prompt length and the largest batch that fits in the KV cache for each group. Bars compare BrowSA at two page sizes and selection ratios with Dense, separating attention, scoring and selection, and other decode work. Each request generates 64 tokens. TPOT is the mean streaming interval, weighted by requests, while all requests in a batch decode together.

chunks of at most 32 tokens. BrowSA achieves the lowest TPOT, 18.26 ms, against 18.89 ms for Quest, 19.21 ms for Block Sparse, and 33.02 ms for Dense. Relative to the sparse baselines, it is faster in both sparse-attention stages. Scoring is 21.0% faster than Quest and 20.4% faster than Block Sparse, which we attribute to mean query pooling (Section 3.1): BrowSA evaluates one envelope per KV head rather than per query head. Attention is 10.2% faster than Quest, 10.4% faster than Block Sparse, and $14.1\times$ faster than Dense. Packing lets BrowSA use page size 64 while selecting small chunks, whereas Quest and Block Sparse must select and fetch at block size 32.

Latency across prompt lengths. Figure 7 measures TPOT on recorded WebVoyager requests grouped by prompt length, with each batch sized so that all of its requests decode together. At selection ratios of 6.25% and 12.5%, BrowSA is faster than Dense in every setting, by up to $1.5\times$ at 10.7K tokens (19.1 vs. 28.2 ms), and stays within 18.2–21.3 ms across all three settings, whereas Dense takes 23.8–28.2 ms. Attention takes 1.1–2.5 ms per decode step for BrowSA, compared with 10.2–14.4 ms for Dense; scoring and page selection add 2.6–4.6 ms, and the remaining decode time of about 14 ms is common to both. Including scoring and selection, BrowSA's attention path is 2.2–$3.1\times$ faster than Dense attention at a 6.25% selection ratio (a 55–68% reduction; 4.6 vs. 14.4 ms per step at 10.7K tokens) and 1.8–$2.5\times$ faster at 12.5%.

Effect of KV cache packing. We compare BrowSA with natural prompt order and with D_{worst} packing under the same selection and attention kernels, with page size 32 on requests from recorded WebVoyager runs. Across four selection ratios from 3.125% to 25%, packing reduces attended pages

Method	GPU time per layer (μs) Scoring	Attention	TPOT (ms)
Dense	0.00	305.43	33.019
BrowSA	18.02	21.65	**18.261**
Quest	22.80	24.12	18.886
Block Sparse	22.65	24.17	19.207

Table 2: Comparing TPOT at matched attended tokens. BrowSA uses a 6.25% selection ratio, Quest and Block Sparse are calibrated to attend the same number of tokens, and Dense attends all tokens.

Selection ratio	Reduction (%) Pages	Attention	TPOT
3.125%	10.06	9.25	0.82
6.25%	11.63	12.41	2.05
12.5%	12.06	14.09	4.49
25%	10.91	12.77	4.53
Max.	**12.06**	**14.09**	**4.53**

Table 3: Effect of GPU packing on WebVoyager. All runs use page size 32, and reductions are relative to natural order.

Variant	Score	Packing	Selection ratio 6.25%	12.5%
BrowSA	BrowSA	✓	**44.4**	**45.6**
w/o packing	BrowSA	–	40.0 (−4.4)	43.3 (−2.3)
w/o packing, Quest score	Quest	–	31.1 (−13.3)	34.4 (−11.2)
Dense (100% KV)	–	–	37.8	

Table 4: **Ablation of the BrowSA score and KV packing on GAIA.** Task success (%) at two selection ratios, using page size 32 and chunk targets of 6 to 16 tokens. Each row removes one more component. Parentheses give the change relative to full BrowSA.

by 10.1–12.1% and pure attention time by 9.3–14.1% (Table 3), and TPOT improves by 4.5% at ratios of 12.5% and above. Appendix A.4 reports the one-time packing cost.

4.3 Ablation Study

We remove KV packing and the calibrated score one at a time on GAIA, at selection ratios of 6.25% and 12.5%, keeping the structural chunks and page size 32 fixed. The first variant leaves chunks in natural order instead of applying D_{worst} packing (Section 3.2), and the second additionally replaces the BrowSA score (Equation 2) with the uncalibrated Quest envelope (Equation 1). The calibrated score contributes most of the gain (Table 4). Replacing the calibrated score lowers success by 8.9 points at both ratios. This drop is consistent with the length inflation measured in Section 3.1: without width calibration, long chunks receive inflated scores. Packing adds a further 4.4 and 2.3 points at the 6.25% and 12.5% selection ratios; the larger gain at the smaller ratio is consistent with co-selected chunks sharing pages, so the same page cap covers more useful context (Section 4.2). The last variant applies Quest's envelope to structural chunks and reaches 31.1% and 34.4%, below Quest on fixed 32-token blocks (32.2% and 38.9%, Figure 6a), so structural chunks help only when the score corrects for their unequal lengths.

5 Conclusion

BrowSA shows that the semantic boundaries already present in browser-agent prompts can decide both what sparse attention selects and how the KV cache is laid out. Structural chunking keeps action targets intact, length-calibrated ranking corrects the score inflation of long chunks, and packing places co-selected chunks on shared physical pages. BrowSA matches or exceeds Dense task success on WebVoyager, GAIA, and Online-Mind2Web at selection ratios where Quest and Block Sparse fall behind, and reduces decode attention latency, including scoring and selection, by up to 68% relative to Dense.

AI USE STATEMENT

In this work, we used generative AI tools to help polish the paper. We have not used generative AI tools to develop theoretical models or conceptual frameworks, propose or refine hypotheses, clean or reformat datasets, or interpret results, and the rest of the required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools to help edit software code, summarize or analyze existing literature, source or search for information. We reviewed all AI-assisted text for accuracy, clarity, and alignment with the work. We checked technical claims, methods, and interpretations against our experiments and analysis; tested and verified AI-assisted code and artifacts for correctness. We did not use AI-generated figures or images. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Our anonymized code is available at `https://anonymous.4open.science/r/BrowserSparseAttention-anonymous-B6FC` to reproduce our design.

REFERENCES

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025. doi: 10.48550/arXiv.2511.21631.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. In *Conference on Language Modeling*, 2025.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. arXiv preprint arXiv:2302.01318, 2023.

Jinhan Chen, Jianchun Liu, Hongli Xu, Xianjun Gao, and Shilong Wang. SABlock: Semantic-aware KV cache eviction with adaptive compression block size. *arXiv preprint arXiv:2510.22556*, 2025a.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024a.

Renze Chen, Zhuofeng Wang, Beiquan Cao, Tong Wu, Size Zheng, Xiuhong Li, Xuechao Wei, Shengen Yan, Meng Li, and Yun Liang. ArkVale: Efficient generative LLM inference with recallable key-value eviction. In *Advances in Neural Information Processing Systems*, volume 37, pp. 113134–113155, 2024b. doi: 10.52202/079017-3595.

Yaoqi Chen, Jinkai Zhang, Baotong Lu, Qianxi Zhang, Chengruidong Zhang, Jing Liu, Jingjia Luo, Di Liu, Huiqiang Jiang, Qi Chen, Bailu Ding, Xiao Yan, Jiawei Jiang, Chen Chen, Mingxing Zhang, Cheng Li, Yuqing Yang, Fan Yang, and Mao Yang. RetroInfer: A vector storage engine for scalable long-context LLM inference. *Proceedings of the VLDB Endowment*, 19(5):1016–1031, 2026a. doi: 10.14778/3796195.3796212.

Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, and Beidi Chen. MagicPIG: LSH sampling for efficient LLM generation. In *International Conference on Learning Representations*, pp. 44169–44190, 2025b.

Zhuoming Chen, Xinrui Zhong, Qilong Feng, Ranajoy Sadhukhan, Yang Zhou, Michael Qizhe Shieh, Zhihao Jia, and Beidi Chen. Vortex: Efficient and programmable sparse attention serving for AI agents. arXiv preprint arXiv:2606.06453, 2026b.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Anirban DasGupta, Soumendra N. Lahiri, and Jordan Stoyanov. Sharp fixed n bounds and asymptotic expansions for the mean and the median of a Gaussian sample maximum, and applications to the Donoho–Jin model. *Statistical Methodology*, 20:40–62, 2014. doi: 10.1016/j.stamet.2014.01.002.

DeepSeek-AI. DeepSeek-V3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Xiang Deng, Prashant Shiralkar, Colin Lockard, Binxuan Huang, and Huan Sun. DOM-LM: Learning generalizable representations for HTML documents. arXiv preprint arXiv:2201.10608, 2022.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems*, volume 36, pp. 28091–28114, 2023. doi: 10.52202/075280-1220.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable are web agents at solving common knowledge work tasks? In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11642–11662. PMLR, 2024.

Gonçalo Duarte, Miguel Couceiro, and Marcos V. Treviso. EntmaxKV: Support-aware decoding for entmax attention. arXiv preprint arXiv:2605.21649, 2026.

Masafumi Enomoto, Ryoma Obara, Haochen Zhang, and Masafumi Oyamada. Read more, think more: Revisiting observation reduction for web agents. arXiv preprint arXiv:2604.01535, 2026.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Ada-KV: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-3773.

Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Peiyuan Zhou, Jiaxing Qi, Junjie Lai, Hayden Kwok-Hay So, Ting Cao, Fan Yang, and Mao Yang. SeerAttention: Self-distilled attention gating for efficient long-context prefilling. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1869.

Shubham Gaur and Ian Lane. Signal-driven observation for long-horizon web agents. In *ICML 2026 Workshop on Failure Modes in Agentic AI*, 2026.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *International Conference on Learning Representations*, pp. 22975–22988, 2024.

Gonzalo Gonzalez-Pumariega, Vincent Tu, Chih-Lun Lee, Jiachen Yang, Ang Li, and Xin Eric Wang. The unreasonable effectiveness of scaling agents for computer use. In *International Conference on Learning Representations (ICLR)*, 2026.

Boyu Gou, Zanming Huang, Yuting Ning, Yu Gu, Michael Lin, Weijian Qi, Andrei Kopanev, Botao Yu, Bernal Jiménez Gutiérrez, Yiheng Shu, Chan Hee Song, Jiaman Wu, Shijie Chen, Hanane Nour Moussa, Tianshu Zhang, Jian Xie, Yifei Li, Tianci Xue, Zeyi Liao, Kai Zhang, Boyuan Zheng, Zhaowei Cai, Viktor Rozgic, Morteza Ziyadi, Huan Sun, and Yu Su. Mind2Web 2: Evaluating agentic search with agent-as-a-judge. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2025.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. WebVoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6864–6890, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.371.

Coleman Richard Charles Hooper, Sehoon Kim, Hiva Mohammadzadeh, Monishwaran Maheswaran, Sebastian Zhao, June Paik, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. Squeezed attention: Accelerating long context length LLM inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 32631–32652. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1568.

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Tiancheng Hu, Zhang Qin, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. EPIC: Efficient position-independent caching for serving large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 24391–24402. PMLR, 2025.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. MInference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention. In *Advances in Neural Information Processing Systems*, volume 37, pp. 52481–52515, 2024. doi: 10.52202/079017-1663.

Sahil Joshi, Prithvi Dixit, Agniva Chowdhury, Anshumali Shrivastava, Joseph E Gonzalez, Ion Stoica, Kumar Krishna Agrawal, and Aditya Desai. Inference time context sparsity: Illusion or opportunity? *arXiv preprint arXiv:2605.24168*, 2026a.

Sahil Joshi, Prithvi Dixit, Agniva Chowdhury, Anshumali Shrivastava, Joseph E Gonzalez, Ion Stoica, Kumar Krishna Agrawal, and Aditya Desai. Inference time context sparsity: Illusion or opportunity? *arXiv preprint arXiv:2605.24168*, 2026b.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A. Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. ACON: Optimizing context compression for long-horizon LLM agents. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Imene Kerboua, Sahar Omidi Shayegan, Megh Thakkar, Xing Han Lù, Massimo Caccia, Véronique Eglin, Alexandre Aussem, Jérémy Espinas, and Alexandre Lacoste. LineRetriever: Planning-aware observation reduction for web agents. arXiv preprint arXiv:2507.00210, 2025.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020.

Donguk Kwon and Dongha Lee. Region4Web: Rethinking observation space granularity for web agents. arXiv preprint arXiv:2605.07134, 2026.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626. Association for Computing Machinery, 2023. doi: 10.1145/3600006.3613165.

Xunhao Lai, Jianqiao Lu, Yao Luo, Yiyuan Ma, and Xun Zhou. FlexPrefill: A context-aware sparse attention mechanism for efficient long-sequence inference. In *International Conference on Learning Representations*, pp. 963–989, 2025.

Thibault Le Sellier de Chezelles, Maxime Gasse, Alexandre Lacoste, Massimo Caccia, Alexandre Drouin, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Graham Neubig, Quentin Cappart, Russ Salakhutdinov, and Nicolas Chapados. The BrowserGym ecosystem for web agent research. *Transactions on Machine Learning Research*, 2025.

Dongjun Lee, Juyong Lee, Kyuyoung Kim, Jihoon Tack, Jinwoo Shin, Yee Whye Teh, and Kimin Lee. Learning to contextualize web pages for enhanced decision making by LLM agents. In *International Conference on Learning Representations*, pp. 81039–81072, 2025.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. InfiniGen: Efficient generative inference of large language models with dynamic KV cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation*, pp. 155–172. USENIX Association, 2024.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 2023.

Dongfang Li, Zixuan Liu, Gang Lin, Baotian Hu, and Min Zhang. LycheeCluster: Efficient long-context inference with structure-aware chunking and hierarchical KV indexing. *arXiv preprint arXiv:2603.08453*, 2026.

Junlong Li, Yiheng Xu, Lei Cui, and Furu Wei. MarkupLM: Pre-training of text and markup language for visually rich document understanding. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6078–6087. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.acl-long.420.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*, volume 37, pp. 22947–22970, 2024. doi: 10.52202/079017-0722.

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. QServe: W4A8KV4 quantization and system co-design for efficient LLM serving. In *Proceedings of Machine Learning and Systems*, volume 7, 2025.

Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, Chen Chen, Fan Yang, Yuqing Yang, and Lili Qiu. RetrievalAttention: Accelerating long-context LLM inference via vector retrieval. In *Advances in Neural Information Processing Systems*, volume 38, 2025a. doi: 10.52202/085713-1816.

Di Liu, Ruitian Wang, Chen Chen, Mingliang Gong, Yongjie Yuan, Han Zhao, Yu Feng, Quan Chen, and Minyi Guo. AB-Sparse: Sparse attention with adaptive block size for accurate and efficient long-context inference. *arXiv preprint arXiv:2605.12110*, 2026.

Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. ClusterKV: Manipulating LLM KV cache in semantic space for recallable compression. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, pp. 1–7, 2025b. doi: 10.1109/DAC63849.2025.11132479.

Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Yue Liu, Bo Li, Xuming Hu, and Xiaowen Chu. ChunkKV: Semantic-preserving KV cache compression for efficient long-context LLM inference. In *Advances in Neural Information Processing Systems*, volume 38, 2025c. doi: 10.52202/085713-0966.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Advances in Neural Information Processing Systems*, volume 36, pp. 52342–52364, 2023. doi: 10.52202/075280-2279.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, Zhiqi Huang, Huan Yuan, Suting Xu, Xinran Xu, Guokun Lai, Yanru Chen, Huabin Zheng, Junjie Yan, Jianlin Su, Yuxin Wu, Yutao Zhang, Zhilin Yang, Xinyu Zhou, Mingxing Zhang, and Jiezhong Qiu. MoBA: Mixture of block attention for long-context LLMs. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0602.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for general AI assistants. In *International Conference on Learning Representations*, pp. 9025–9049, 2024.

Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378*, 2011.

Piotr Nawrot, Jianing Li, Renjie Huang, Sebastian Ruder, Kelly Marchisio, and Edoardo Ponti. The sparse frontier: Sparse attention trade-offs in transformer LLMs. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 38667–38701. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1926.

NVIDIA. TensorRT-LLM. Open-source software library, 2023.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In *International Conference on Learning Representations*, pp. 34433–34448, 2025.

Zaifeng Pan, Ajjkumar Dahyalal Patel, Yipeng Shen, Zhengding Hu, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. KVFlow: Efficient prefix caching for accelerating LLM-based multi-agent workflows. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-4208.

Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, and Ashish Panwar. vAttention: Dynamic memory management for serving LLMs without PagedAttention. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pp. 1133–1150. Association for Computing Machinery, 2025. doi: 10.1145/3669940.3707256.

Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. SparQ attention: Bandwidth-efficient LLM inference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 42558–42583. PMLR, 2024.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021. doi: 10.1162/tacl_a_00353.

Thassilo M. Schiepanski and Nicholas Piël. Beyond pixels: Exploring DOM downsampling for LLM-based web agents. arXiv preprint arXiv:2508.04412, 2025. Version 4, revised August 2026.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. ShadowKV: KV cache in shadows for high-throughput long-context LLM inference. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 57355–57373. PMLR, 2025.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 47901–47911. PMLR, 2024.

Qifan Wang, Yi Fang, Anirudh Ravula, Fuli Feng, Xiaojun Quan, and Dongfang Liu. WebFormer: The web-page transformer for structure information extraction. In *Proceedings of the ACM Web Conference 2022*, pp. 3124–3133. Association for Computing Machinery, 2022. doi: 10.1145/3485447.3512032.

Yanli Wang, Yanlin Wang, Bowen Zhang, Yiwei Zhang, Daya Guo, Jiachi Chen, Hongyu Zhang, and Zibin Zheng. From logical to computational sparsity: Structure-aware block-sparse attention for long-code completion. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 25927–25942, San Diego, California, United States, July 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.1189.

Zihao Wang, Bin Cui, and Shaoduo Gan. SqueezeAttention: 2D management of KV-cache in LLM inference via layer-wise optimal budget. In *International Conference on Learning Representations*, pp. 23593–23606, 2025.

Mike Wong, Kevin Hsieh, Suman Nath, and Ravi Netravali. Skim: Speculative execution for fast and efficient web agents. arXiv preprint arXiv:2605.16565, 2026.

Fang Wu, Congming Gao, Weixi Zhu, and Jiwu Shu. PRKV: Page restruct KV cache for high accuracy and efficiency LLM generation. OpenReview manuscript, 2025. Submitted to ICLR 2026.

Shunlong Wu, Hai Lin, Shaoshen Chen, Tingwei Lu, Yongqin Zeng, Shaoxiong Zhan, Hai-Tao Zheng, and Hong-Gee Kim. SemantiCache: Efficient KV cache compression via semantic chunking and clustered merging. *arXiv preprint arXiv:2603.14303*, 2026.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, pp. 21875–21895, 2024.

Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. XAttention: Block sparse attention with antidiagonal scoring. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 69819–69831. PMLR, 2025.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? Assessing the current state of web agents. In *Conference on Language Modeling*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a. doi: 10.48550/arXiv.2505.09388.

Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor, Pratik A. Chaudhari, George Karypis, and Huzefa Rangwala. AgentOccam: A simple yet strong baseline for LLM-based web agents. In *International Conference on Learning Representations*, pp. 97533–97565, 2025b.

Lijie Yang, Zhihao Zhang, Zhuofu Chen, Zikun Li, and Zhihao Jia. TidalDecode: Fast and accurate LLM decoding with position persistent sparse attention. In *International Conference on Learning Representations*, pp. 5823–5844, 2025c.

Lijie Yang, Zhihao Zhang, Arti Jain, Shijie Cao, Baihong Yuan, Yiwei Chen, Zhihao Jia, and Ravi Netravali. Less is more: Fast and accurate reasoning with cross-head unified sparse attention. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Shang Yang, Junxian Guo, Haotian Tang, Qinghao Hu, Guangxuan Xiao, Jiaming Tang, Yujun Lin, Zhijian Liu, Yao Lu, and Song Han. LServe: Efficient long-sequence LLM serving with unified sparse attention. In *Proceedings of Machine Learning and Systems*, volume 7, 2025d.

Shuo Yang, Ying Sheng, Joseph E. Gonzalez, Ion Stoica, and Lianmin Zheng. Post-training sparse attention with double sparsity. *arXiv preprint arXiv:2408.07092*, 2024.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 94–109, 2025. doi: 10.1145/3689031.3696098.

Jiancai Ye, Jun Liu, Qingchen Li, Tianlang Zhao, Hanbin Zhang, Jiayi Pan, Ningyi Xu, and Guohao Dai. DynSplit-KV: Dynamic semantic splitting for KVCache compression in efficient long-context LLM inference. *arXiv preprint arXiv:2602.03184*, 2026.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. FlashInfer: Efficient and customizable attention engine for LLM inference serving. In *Proceedings of Machine Learning and Systems*, volume 7, 2025.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23078–23097. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1126.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big Bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems*, volume 33, pp. 17283–17297, 2020.

Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. PQCache: Product quantization-based KVCache for long context LLM inference. *Proceedings of the ACM on Management of Data*, 3(3):201:1–201:30, 2025. doi: 10.1145/3725338.

Jiayuan Zhang, Kaiquan Chen, Zhihao Lu, Enshen Zhou, Qian Yu, and Jing Zhang. Prune4Web: DOM tree pruning programming for web agent. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(41):34710–34718, 2026. doi: 10.1609/aaai.v40i41.40772.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang "Atlas" Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 34661–34710, 2023. doi: 10.52202/075280-1506.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583, 2024. doi: 10.52202/079017-2000.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*, pp. 15585–15606, 2024.

Kan Zhu, Tian Tang, Qinyu Xu, Zhan Jin, Yile Gu, Zhichen Zeng, Rohan Kadekodi, Liangyu Zhao, Ang Li, Arvind Krishnamurthy, and Baris Kasikci. Tactic: Adaptive sparse attention with clustering and distribution fitting for long-context LLMs. In *International Conference on Learning Representations*, pp. 30626–30644, 2026.

Yuxuan Zhu, Ali Falahati, David H. Yang, and Mohammad Mohammadi Amiri. SentenceKV: Efficient LLM inference via sentence-level semantic KV caching. In *Conference on Language Modeling*, 2025.

A Appendix

A.1 Expected envelope slack under Gaussian keys

Proposition 1 (Expected Gaussian envelope gap)**.** *Let $k_1, \dots, k_L$ be independent draws from $\mathcal{N}(\mu, \Sigma)$, where Σ is positive semidefinite, and fix q independently of the keys. Define the envelope $U_L = \sum_i \max_{t \le L} q_i k_{t,i}$, actual maximum $T_L = \max_{t \le L} q^\top k_t$, and gap $G_L = U_L - T_L$. For $a_L = \mathbb{E} \max_{t \le L} Z_t$ with independent standard normal Z_t,*

$$\mathbb{E}G_L = \kappa(q, \Sigma) a_L, \qquad \kappa(q, \Sigma) = \sum_i |q_i| \sqrt{\Sigma_{ii}} - \sqrt{q^\top \Sigma q} \ge 0. \tag{6}$$

When $\kappa > 0$, the expected gap is asymptotic to $\kappa\sqrt{2 \ln L}$.

Under the assumptions of Proposition 1, each coordinate $q_i k_{t,i}$ and each projection $q^\top k_t$ is Gaussian. Standardizing the nondegenerate variables and treating zero variance terms as constants gives

$$\mathbb{E}U_L = q^\top \mu + a_L \sum_i |q_i| \sqrt{\Sigma_{ii}}, \qquad \mathbb{E}T_L = q^\top \mu + a_L \sqrt{q^\top \Sigma q}. \tag{7}$$

The first equality uses linearity of expectation, so it does not require independent coordinates within a key. Subtracting the two expressions proves Equation 6. Nonnegativity follows from the triangle inequality $\sqrt{q^\top \Sigma q} \le \sum_i |q_i| \sqrt{\Sigma_{ii}}$, applied to the centered coordinate variables in L^2. The classical result $a_L \sim \sqrt{2 \ln L}$ gives the stated asymptotic rate (DasGupta et al., 2014). For $\Sigma = \sigma^2 I$, the coefficient reduces to $\sigma(\|q\|_1 - \|q\|_2)$. A common attention scale $1/\sqrt{d}$ multiplies both expressions and their difference by that factor.

Scope. This calculation applies established Gaussian maximum results to Quest's envelope (Tang et al., 2024). Independence across keys and independence of the query from those keys are analytical assumptions, not established properties of our traces. The result concerns the additive score gap, without a guarantee for relative error, attention mass, or model outputs. Neither its asymptotic rate nor the finite range empirical fit establishes a universal length law or derives $\alpha = 0.25$.

A.2 Score diagnostic protocol and finite range fits

Captures and sampling. The archive contains 23 captures representing 21 unique token sequences from 13 selected tasks, including two GAIA tasks and 11 WebVoyager tasks (Mialon et al., 2024; He et al., 2024). The captures were selected from action disagreements between a sparse configuration and dense reference decoding, rather than sampled to represent deployment traffic. We use Qwen3-VL-30B-A3B-Instruct layers 0, 12, 24, and 36, and the first, fifth, and last recorded selector calls in each capture (Bai et al., 2025). Queries come from the captured sparse trajectory and are evaluated against its stored prefill keys. Newly generated keys are excluded. For each layer and each of three random seeds, 64 starting positions are sampled with replacement from 0 through $N - 1024$, inclusive, where N is the prefill length. This supports the full audit's maximum window length of 1,024. Figure 5 uses the nested lengths 8, 16, 32, 64, 128, and 256 at these same sampled starts. The windows may contain any prefill context role and do not follow semantic chunk boundaries.

Scores and aggregation. The envelope center is a box midpoint score, not the score of the mean key. Its width alone is not estimation error because the actual maximum T can also grow. Each of the four KV heads is shared by eight query heads. We first average those eight queries, then compute U, T, c, w, and G independently for each KV head and average the resulting scalars. In particular, T is the mean of per-group maxima, not the maximum of a mean across groups. The resulting envelope bounds the pooled query, not every constituent query head. Keys are stored in half precision, scores are computed in single precision, and all measured logits include the $1/\sqrt{128}$ attention scale. Window and query observations are averaged within each layer and seed cell. We average those cells within captures, repeated captures within identical token sequences, and unique prompts within tasks. The source archive averages all 13 tasks equally, while Figure 5 averages only the five evaluation tasks defined below. The pointwise 95% intervals use 5,000 paired task bootstrap resamples drawn from the corresponding task set. These intervals quantify variation among the selected tasks, without a population representativeness claim or correction for comparisons across lengths.

Calibrated score comparison. We fix the reference length at one token and fit the exponent to task summaries using all six measured lengths, without filtering lengths by the resulting error. We sort task names, permute their indices with NumPy's default random generator, and assign the first eight tasks to fitting and the remaining five to evaluation. The evaluation tasks are BBC News–41, level1-6, Allrecipes–25, ESPN–33, and GitHub–22. Within the fitting tasks, we minimize the mean squared residual between each task's mean calibrated score and mean exact maximum, giving equal weight to tasks and lengths. Searching $\alpha \in [0, 1]$ in increments of 10^{-4} gives $\alpha = 0.397$. The fitting routine receives only fitting-task summaries. This is a retrospective split of the selected archive, not a new external test set. These diagnostic parameters differ from the serving settings in Section 4.

Both panels use the same five evaluation tasks, recorded queries, keys, and exact maxima. For each task and length, we compute $s = c + L^{-0.397}w$ from the original summaries. Since the multiplier is constant at a fixed length, this gives the same mean score as applying the formula before averaging. Across the 30 evaluation task and length pairs, the mean absolute residual falls from 14.1055 for U to 1.5478 for s, an 89.03% reduction. This metric averages $|\bar{s}_{i,L} - \bar{T}_{i,L}|$ over tasks i and lengths L, where bars denote within-task means. It does not measure per-window absolute error, which can be larger because averaging permits cancellation. The mean signed residuals at lengths 8, 16, 32, 64, 128, and 256 are 2.5159, 1.8133, 0.8507, -0.2514, -1.3816, and -2.4583. Thus calibration reduces the absolute residual of task means but underestimates long windows on average.

Yellow regions mark positive differences between the plotted mean curves, and gray hatching marks negative differences. These areas are not confidence intervals. In panel (b), differences of plotted means need not equal the mean absolute residual, whereas $U \geq T$ makes them equal in panel (a). We jointly bootstrap the five evaluation tasks, keeping the fitted parameters fixed, and export the conditional score and residual intervals with the plotting data. These intervals do not include uncertainty from parameter fitting. The comparison recomputes scores on archived windows and does not establish improved chunk selection, attention preservation, or task success.

Descriptive fits. Across all 13 source tasks and nested windows from 8 to 256 tokens, the task averaged gap rises from 8.27 to 19.59 scaled logits, while the actual maximum rises from 1.92 to 5.22. These measurements compare the same starting positions, queries, and layers across lengths. Ordinary least squares on the six original gap means gives $\widehat{G}(L) = 1.6669 + 3.2556 \ln L$, with $R^2 = 0.99920$ and maximum relative residual 1.96%. An affine fit against $\sqrt{\ln L}$ also fits closely, with $R^2 = 0.99822$. These six points therefore do not identify an asymptotic law. The width has its own descriptive fit, $\widehat{w}(L) = 1.7493 + 4.1697 \ln L$, which must not be substituted for the gap. A power fit to the width over the same range gives an exponent near 0.25, but local exponents vary with length. This descriptive width fit does not derive the diagnostic exponent or prove that the calibrated ranking is optimal.

A.3 Ranking and protected context

Calibration and query aggregation. Earlier work also improves ranking by reducing geometric summary radii (Chen et al., 2024b). Our correction ties the width weight to the unequal lengths introduced by semantic chunking. The score recovers the unmodified envelope at the reference length A. The reported serving configuration uses $A = 32$ and $\alpha = 0.25$. These are empirical settings, not a consequence of Proposition 1. For $L_j > A$ and $\alpha > 0$, the calibrated score need not upper bound the contained scores. It ranks chunks for selection under the configured selection ratio. Each KV head ranks chunks with its pooled group query, so a selected set is shared only by the query heads of one GQA group. Although $\alpha = 0$ recovers the envelope formula on pooled queries, it does not reproduce Quest's original selection policy for individual heads.

Protected context. Prompt chunks follow the structural partition in Section 3.1. An explicit protected set is included independently of ranking. The default policy retains the first page, the last two pages, and all tokens generated within the request. These overlapping spans are combined by union and counted inside the selection ratio's page cap. Protection depends on sequence position and the request boundary, rather than a fraction allocated to a context role. Comparisons must hold protection and admission policies fixed to isolate the effects of chunk geometry and scoring.

Algorithm 1 BrowSA from prompt setup to sparse decoding

Input: Prompt X, chunk limits, A, α, selection ratio, page capacity P, boundary policy, retention rule G_t

Partition X into $\mathcal{C} = \{C_j\}$ ▷ Sec. 3.1
Prefill X to obtain K, V and sample x
Cache b_j, u_j and compute pairwise D_{worst} for each (ℓ, h) ▷ Eqs. (4), (5)
Pack K, V by average linkage, fix boundaries, record $\pi_{\ell,h}$ ▷ Sec. 3.2
Emit x, $t \leftarrow 1$
while $x \neq$ EOS and the output limit is not reached **do**
  for each layer ℓ in model order, processing x **do**
    Form Q, append new K, V, update $\pi_{\ell,h}$
    Compute $\bar{q}_h$ and $s_{j,h}$ for each KV head h ▷ Eqs. (3), (2)
    Admit chunk indices $\mathcal{S}_h$ in descending $s_{j,h}$ within the selection ratio, $\mathcal{T}_h \leftarrow (\bigcup_{j \in \mathcal{S}_h} C_j) \cup G_t$ ▷ Sec. 3.1
    Fetch $\{\pi_{\ell,h}(i) : i \in \mathcal{T}_h\}$ for each h, attend valid causal entries with Q, finish layer
  end for
  Sample and emit x, $t \leftarrow t + 1$
end while

A.4 Placement construction and execution details

Execution and selection policies. Algorithm 1 uses the chunk ranking and selection-ratio admission of Section 3.1. Chunk boundaries are computed from the serialized prompt before prefill. Prefill supplies prompt keys, values, and the logits from which the first output token is sampled. Packing then constructs a fixed layout independently for each layer and KV head, using only prompt key summaries and chunk lengths. Placement constructs an average-linkage tree, orients its leaves, and applies the same token permutation to keys and values. It records their logical-to-page map while keeping the configured boundary chunks at their original offsets. At each subsequent decoding step, the last emitted token passes through the layers in order. Each layer appends that token's key and value before sparse attention, preserving its logical position. This append extends the layer's page maps for every KV head. The generated suffix is always retained and is not inserted into the fixed prompt hierarchy. Scoring pools the queries of each GQA group and computes the calibrated score separately for each KV head, using the stored length weights in u. Here $S_{\ell,h}$ denotes Equation 4 evaluated with the stored summaries of layer ℓ and KV head h. The token-level union of admitted chunks and protected positions is then mapped to distinct physical pages for each head. Page attention uses the original query heads and all valid, causally visible entries on those pages, including any unselected neighbors. The backend uses logical positions for causal visibility and preserves the cached positional encodings.

How the selection ratio is applied is specified for each experiment, rather than fixed by the placement algorithm. Protection is a union of logical token positions, not a union of chunk IDs. The packing ablation in Section 4.2 applies the selection ratio to prompt tokens, admits ranked chunks, and then adds mandatory chunks and generated history. The latency comparison in Section 4.2 instead counts the distinct pages of mandatory entries inside the page cap set by the selection ratio, then admits an ordinary chunk only if its additional pages fit. Mandatory entries remain attended even when they exceed that cap. Both experiments protect whole chunks overlapping the first 64 original tokens and the latest 128 sequence tokens, which at page size 64 are the first page and last two pages of Appendix A.3. These variants share the same per-KV-head ranking, placement, and per-step execution order but differ in admission and retention. Their measured outputs and costs should not be attributed to an identical selected token set.

Scope and key summaries. Each transformer layer and KV head has its own keys, values, and physical permutation, with shared logical prompt chunk boundaries. Let N denote the number of nonempty prompt chunks. Equation 4 suppresses the layer and head indices. Here $b_j \in \mathbb{R}^d$ is the bounding-box midpoint, not the arithmetic mean of token keys; $u_j \in \mathbb{R}^d$ is the length-corrected half-width, not a standard deviation. The reported serving configuration uses $A = 32$ and $\alpha = 0.25$. Absolute values on vectors are coordinate-wise. S is the per-head calibrated score $s_{j,h}$ of Equation 2. Unlike a step-0 score feature, (b_j, u_j) depends only on the chunk's keys and length.

Meaning of the distance. Appendix A.5 proves the exact identity

$$D_{\text{worst}}(C_a, C_b) = \sup_{q \neq 0} \frac{|S(q, C_a) - S(q, C_b)|}{\|q\|_2}. \tag{8}$$

Consequently, every actual query, including one with norm greater than one, satisfies $|S(q, C_a) - S(q, C_b)| \leq \|q\|_2 D_{\text{worst}}(C_a, C_b)$. Small distance therefore means similar calibrated scores across query directions, after accounting for query magnitude. It does not imply that either chunk is hot, that their dense-attention masses are similar, or that a ratio-limited selector always admits them together. Their proximity is a placement heuristic, not a page-saving guarantee.

Average-linkage tree construction. We compute the $N(N-1)/2$ distinct pair distances and apply unweighted pair-group method with arithmetic mean (UPGMA), also called average-linkage agglomerative clustering (Müllner, 2011). For two disjoint groups $\mathcal{U}, \mathcal{V}$, their linkage distance is

$$\Delta(\mathcal{U}, \mathcal{V}) = \frac{1}{|\mathcal{U}||\mathcal{V}|} \sum_{C_a \in \mathcal{U}} \sum_{C_b \in \mathcal{V}} D_{\text{worst}}(C_a, C_b). \tag{9}$$

For a third group $\mathcal{W}$, the merged distance can be updated without recomputing key-vector distances:

$$\Delta(\mathcal{U} \cup \mathcal{V}, \mathcal{W}) = \frac{|\mathcal{U}|\Delta(\mathcal{U}, \mathcal{W}) + |\mathcal{V}|\Delta(\mathcal{V}, \mathcal{W})}{|\mathcal{U}| + |\mathcal{V}|}. \tag{10}$$

Here, $|\mathcal{U}|$ and $|\mathcal{V}|$ denote the numbers of semantic chunks in groups $\mathcal{U}$ and $\mathcal{V}$, respectively; the weights count chunks, not tokens. In particular, this is an average of pair distances, not a distance between group-mean keys. After $N-1$ merges, the result is a binary tree with chunks as leaves; no number of clusters or page-capacity constraint is imposed. UPGMA accepts the pairwise distance directly and does not require the synthetic-center optimization used by k-means. Arithmetic-mean center updates are generally not minimizers of a squared-D_{worst} objective.

Tree orientation and chunk order. At every internal node, we place the child subtree containing the smaller original chunk index first, then traverse leaves left to right. This produces a permutation $\sigma = (\sigma_1, \ldots, \sigma_N)$. It preserves each merged subtree as a contiguous run, but does not minimize the total distance between adjacent leaves. The orientation rule resolves left/right choices *after* merging; equal-distance merge choices are left to the linkage implementation. Reproducibility therefore requires fixed input order, numerical precision, and linkage implementation, not just this orientation rule. For a single chunk, the permutation is the identity.

Boundary-preserving placement. The serving implementation fixes the original offsets of whole chunks overlapping the first 64 and last 128 prompt tokens. Only the remaining middle chunks enter the distance matrix and linkage tree. Their reordered tokens fill the original middle interval without padding. Equation 11 includes the interval's global start offset o_{base}, while fixed chunks retain their original mappings. With zero or one movable chunk, no clustering is needed. The unrestricted construction, including Figure 4, is the special case without fixed boundary chunks. These fixed prompt boundaries determine placement, whereas the recent-token retention window continues to advance during decoding.

From the permutation to physical pages. Let P denote page capacity in token slots ($P = 64$ in the packing analysis). For reordered chunks, let o_{base} be the start of the movable interval in the cache ($o_{\text{base}} = 0$ for unrestricted placement). The global start offset of chunk C_{σ_t} is

$$o_{\sigma_t} = o_{\text{base}} + \sum_{r<t} L_{\sigma_r}, \qquad \text{slot}(C_j, v) = o_j + v, \qquad \text{page}(C_j, v) = \left\lfloor \frac{o_j + v}{P} \right\rfloor, \tag{11}$$

where $v \in \{0, \ldots, L_j - 1\}$ is a token's offset within its chunk. Tokens retain their order within each chunk. There is no inter-chunk padding, page-alignment step, or search for a chunk that fits the remaining space. A cluster can span pages and a chunk can cross a page boundary. In Figure 4b, C_1, C_3, and C_5 have lengths 40, 8, and 40. The first packed page contains C_1, C_3, and 16 tokens of C_5, whose remaining 24 tokens continue on the next page. The algorithm does not change this order to fill the page with a different chunk.

All $T = \sum_j L_j$ prompt tokens remain stored, requiring $\lceil T/P \rceil$ pages per head regardless of the permutation, apart from backend allocation details. The objective is to reduce the number of pages *fetched for a selection*, not the number allocated for the prompt. Figure 4b applies this construction to the same chunks used for ranking in panel (a). The selected set touches three pages in the original order and two in the packed order, before any protected or generated tokens are included. The example demonstrates page accounting rather than a measured latency gain.

For selected chunk indices $\mathcal{S}_{\ell,h}$, let $\mathcal{T}_{\ell,h} = \bigcup_{j \in \mathcal{S}_{\ell,h}} C_j$ and let G contain protected logical positions. If $\pi_{\ell,h}(i)$ maps logical token i to its physical page in this layer/head layout, define

$$\begin{aligned} N_{\mathrm{sel}}(\ell,h) &= |\mathcal{T}_{\ell,h}|, \\ F_{\mathrm{page}}(\ell,h) &= |\{\pi_{\ell,h}(i) : i \in \mathcal{T}_{\ell,h} \cup G\}|, \\ N_{\mathrm{page}}(\ell,h) &= P F_{\mathrm{page}}(\ell,h). \end{aligned} \tag{12}$$

F_{page} counts distinct fetched pages; N_{page} counts their token-slot capacity, not pages. They are traffic proxies for a backend that loads whole pages. The per-KV-head selector of Section 3.1 supplies a distinct $\mathcal{S}_{\ell,h}$ for each head; a shared-head selector would be the special case $\mathcal{S}_{\ell,h} = \mathcal{S}_\ell$ for all h. Independent layouts do not themselves change the selection policy.

Construction cost and execution boundary. For one layer/head, constructing extrema costs $O(Td)$, all pair distances cost $O(N^2 d)$, and the nearest-neighbor-chain implementation of average linkage costs $O(N^2)$ time and memory after distances are available (Müllner, 2011). Tree orientation and chunk offsets take $O(N)$ work; materializing the token permutation takes $O(T)$. Thus layout construction costs $O(Td + N^2 d + N^2)$, excluding physical K/V copying. The one-time packing overhead for WebVoyager is approximately 29 ms per request, measured with page size 64 at batch size 32. In the same runs, packing lowers TPOT by 0.45–0.70 ms, so the overhead is recovered after about 40–63 output tokens.

The layout builder returns head-specific logical-row permutations and chunk offsets. The serving backend applies matching permutations to keys and values and supports head-specific addressing. It must preserve original positional encodings and derive causal visibility from logical positions, not reordered slot indices. Unselected neighbors on fetched pages must remain masked if the attended set is to stay unchanged. Algorithm 1 instead makes page expansion explicit, as in the serving measurements, so valid neighbors may also receive attention. The prompt permutation is fixed across decode steps; generated entries require a separate append/addressing policy and are not reclustered by this prompt-only procedure. Construction, transfer, K/V copying, and address indirection must be amortized over subsequent decoding. Offline page-fetch savings alone establish neither end-to-end TPOT gains nor an optimal physical layout.

A.5 Exact worst-case distance for calibrated chunk scores

We prove the closed form used in Section 3.2. The statement concerns the calibrated chunk score, not dense softmax attention or a guarantee about page fetches. In particular, it does not require an observed query to have Euclidean norm at most one.

Setup and notation. Fix one request, transformer layer, and KV head. A chunk C_j contains L_j key vectors $k_{j,t} \in \mathbb{R}^d$, with token index $t \in \{1, \dots, L_j\}$. For coordinate $i \in \{1, \dots, d\}$, define

$$\begin{aligned} M_{j,i} &= \max_t k_{j,t,i}, \qquad m_{j,i} = \min_t k_{j,t,i}, \\ b_{j,i} &= \frac{M_{j,i} + m_{j,i}}{2}, \quad u_{j,i} = \left(\frac{A}{L_j}\right)^{\alpha} \frac{M_{j,i} - m_{j,i}}{2}. \end{aligned} \tag{13}$$

Here $A > 0$ is the length anchor and α is the fixed calibration exponent from Section 3.1. The vector b_j is a bounding-box midpoint, whereas the lowercase m_j denotes coordinate minima as in the main text. The vector u_j is a corrected half-width. Neither is a token-key mean or standard deviation. The proof applies to any fixed choices of these summaries, including other fixed length-correction factors.

For a query $q \in \mathbb{R}^d$, write

$$S(q, C_j) = \sum_{i=1}^{d} (q_i b_{j,i} + |q_i| u_{j,i}). \tag{14}$$

This is the per-head score before head aggregation. As in the main text, the common attention scaling factor is omitted. For two chunks C_a, C_b, introduce

$$\begin{aligned} x &= b_a - b_b, & y &= u_a - u_b, \\ p_i &= \max(|x_i| + y_i, 0), & n_i &= \max(|x_i| - y_i, 0), \\ D_{\mathrm{worst}}(C_a, C_b) &= \max\{\|p\|_2, \|n\|_2\}. \end{aligned} \tag{15}$$

Thus x measures midpoint differences and y measures corrected-width differences. The nonnegative vectors p and n account for the two signed score gaps. The symbol n is a vector, not the number of chunks. For any vectors v, w, $v^\top w = \sum_i v_i w_i$ and $\|v\|_2 = (\sum_i v_i^2)^{1/2}$.

Proposition (exact score difference per unit query norm). For every $q \in \mathbb{R}^d$,

$$|S(q, C_a) - S(q, C_b)| \le \|q\|_2 D_{\mathrm{worst}}(C_a, C_b). \tag{16}$$

Moreover, for every radius $R \ge 0$,

$$\max_{\|q\|_2 \le R} |S(q, C_a) - S(q, C_b)| = R D_{\mathrm{worst}}(C_a, C_b). \tag{17}$$

Consequently, D_{worst} is the smallest coefficient for which Equation (16) holds for all queries, and equals the supremum in Equation (8).

Proof, step 1: expand the score gap. Let $g(q) = S(q, C_a) - S(q, C_b)$, $r = |q|$, and $R_q = \|q\|_2 = \|r\|_2$. In particular, $r_i \ge 0$, and R_q is the actual query magnitude, with no upper restriction. Subtracting the scores gives

$$g(q) = q^\top x + r^\top y = \sum_i (q_i x_i + r_i y_i). \tag{18}$$

Step 2: bound the positive gap. For each coordinate, $q_i x_i \le |q_i x_i| = |q_i||x_i| = r_i|x_i|$. Hence

$$\begin{aligned} g(q) &\le \sum_i r_i (|x_i| + y_i) \\ &\le \sum_i r_i \max(|x_i| + y_i, 0) = r^\top p \\ &\le \|r\|_2 \|p\|_2 = R_q \|p\|_2. \end{aligned} \tag{19}$$

The second inequality uses $r_i \ge 0$: replacing a negative coefficient by zero cannot decrease the sum. It does not remove terms from the actual score; it constructs an upper bound. The final step is the Cauchy–Schwarz inequality $|v^\top w| \le (\sum_i v_i^2)^{1/2} (\sum_i w_i^2)^{1/2}$, applied to $v = r, w = p$. Since r, p are nonnegative, their dot product is also nonnegative.

Step 3: bound the negative gap. The opposite signed gap is $-g(q) = S(q, C_b) - S(q, C_a)$. Coordinate-wise,

$$-q_i x_i \le |q_i x_i| = |q_i||x_i| = r_i |x_i|. \tag{20}$$

Subtracting the same $r_i y_i$ from both sides and summing yields

$$\begin{aligned} -g(q) &= \sum_i (-q_i x_i - r_i y_i) \\ &\le \sum_i r_i (|x_i| - y_i) \\ &\le \sum_i r_i \max(|x_i| - y_i, 0) = r^\top n \\ &\le \|r\|_2 \|n\|_2 = R_q \|n\|_2. \end{aligned} \tag{21}$$

Here $r^\top n = \sum_i |q_i| n_i$ is the dot product with the nonnegative coefficient vector n, not an extra score used by the selector. It bounds the advantage of C_b over C_a. Again, positive-part clipping uses $r_i \ge 0$, and the last inequality is Cauchy–Schwarz.

Step 4: combine the signs. For any real scalar z, $|z| = \max(z, -z)$. Therefore

$$|g(q)| \le R_q \max\{\|p\|_2, \|n\|_2\} = \|q\|_2 D_{\text{worst}}(C_a, C_b), \tag{22}$$

which proves Equation (16) and the upper-bound direction of Equation (17).

Step 5: construct queries attaining equality. Define $s_i = 1$ if $x_i \ge 0$ and $s_i = -1$ otherwise, so $s_i x_i = |x_i|$, including at $x_i = 0$. Suppose first that $\|p\|_2 \ge \|n\|_2$ and $p \ne 0$. For any $R > 0$, choose

$$r_i^* = \frac{R p_i}{\|p\|_2}, \qquad q_i^* = s_i r_i^*. \tag{23}$$

Then $\|q^*\|_2 = R$. Coordinates where $p_i = 0$ receive zero magnitude; elsewhere $|x_i| + y_i = p_i$. Thus

$$g(q^*) = \sum_i r_i^*(|x_i| + y_i) = \frac{R}{\|p\|_2} \sum_i p_i^2 = R\|p\|_2 = R D_{\text{worst}}(C_a, C_b). \tag{24}$$

This also explains why clipping is exact for the maximization: a maximizing query can put zero magnitude on coordinates with negative coefficients. Cauchy–Schwarz is tight because r^* is proportional to p.

If instead $\|n\|_2 > \|p\|_2$, choose

$$r_i^* = \frac{R n_i}{\|n\|_2}, \qquad q_i^* = -s_i r_i^*. \tag{25}$$

The signs now oppose x, and the same calculation gives

$$-g(q^*) = \sum_i r_i^*(|x_i| - y_i) = R\|n\|_2 = R D_{\text{worst}}(C_a, C_b). \tag{26}$$

If $p = n = 0$, then $|x_i| + y_i \le 0$ and $|x_i| - y_i \le 0$ for every i. Adding these inequalities gives $x_i = 0$, and then $y_i = 0$. The scores agree for all queries. If $R = 0$, the only feasible query is zero and both sides of Equation (17) vanish. These cases complete the proof. □

Example with a non-unit query. Let $x = (1, -2)$ and $y = (3, -1)$. These differences can arise, for example, from $b_a = (1, -2)$, $b_b = (0, 0)$, $u_a = (3, 0)$, and $u_b = (0, 1)$. Then $p = (4, 1)$, $n = (0, 3)$, and $D_{\text{worst}} = \sqrt{17}$. Taking

$$q = \left(\frac{20}{\sqrt{17}}, -\frac{5}{\sqrt{17}}\right) \quad \text{gives} \quad \|q\|_2 = 5, \qquad g(q) = \frac{30 + 55}{\sqrt{17}} = 5\sqrt{17}. \tag{27}$$

Equality holds with query norm five. The unit-radius formulation is a way of factoring out magnitude, not an assumption that recorded queries are normalized.

Relation to the simpler upper bound. By the triangle inequality and Cauchy–Schwarz,

$$|q^\top x + |q|^\top y| \le \|q\|_2(\|x\|_2 + \|y\|_2). \tag{28}$$

Consequently $D_{\text{worst}} \le D_{\text{bound}}$, where $D_{\text{bound}} = \|x\|_2 + \|y\|_2$. The closed form above accounts for the coupled query signs and coordinate magnitudes and is the exact coefficient for the specified score, whereas this sum-of-norms bound can be loose. The use of L2 corresponds to measuring query magnitude in Euclidean norm; a different query-norm constraint would yield a different optimization problem. Restoring the usual factor $1/\sqrt{d}$ in both scores scales the bound by the same factor.

What the result does not establish. The theorem holds for individual query vectors, including pooled queries, within one layer/head. It neither assumes nor predicts their distribution. A small distance does not imply large scores, identical selected sets, similar dense-attention probabilities, or fewer fetched pages. Distinct key sets can also have the same summaries and hence zero distance. UPGMA and leaf ordering remain heuristic choices for physical placement; their page savings and construction overhead require separate evaluation.

A.6 CONTEXT OBSERVATIONS AND EXAMPLES

Constructed overview. Figure 4 uses assigned token counts and one-dimensional key summaries for one KV head. The example uses a 64-token chunk cap, a six-token merge target, reference length $A = 32$, and exponent $\alpha = 0.25$. The five chunks cover $[0, 40)$, $[40, 60)$, $[60, 68)$, $[68, 132)$, and $[132, 172)$. Four opening form tokens join C_1, and four closing tokens join C_5. Their box midpoints are $b = (8, 0, 10, 0.2, 10.1)$ and their calibrated half-widths are $u_j = 0.5$. These summaries are realized by extrema $b_j \pm 0.5(L_j/32)^{0.25}$, with all remaining keys at b_j. The resulting distance is $D_{\text{worst}}(C_a, C_b) = |b_a - b_b|$. Average linkage and the minimum-index orientation produce the fixed order $(C_1, C_3, C_5, C_2, C_4)$ without observing a query. A later query $q = 1$ assigns scores $(8.5, 0.5, 10.5, 0.7, 10.6)$ and selects the top two chunks, C_5 and C_3, containing 48 tokens in total. With 64 slots per page, these chunks touch three pages in the original layout and two after reordering. Both layouts retain all 172 tokens in three allocated pages. The example excludes independently protected and generated tokens, whose union can change the fetched page count. It does not imply that the serving evaluation attends exactly these 48 tokens or that every selection saves a page.

Boundary examples. The six Amazon options in Figure 2 occupy 187 tokens. Option boundaries give groups of 42, 30, 30, 29, 29, and 27 tokens, with the listbox prefix attached to the first. The illustrative 16-token grid starts at the beginning of this isolated DOM subtree, rather than at the beginning of the full model input. It splits target option 25926 into fragments of 16 and 11 tokens. The next boundary falls five tokens beyond the option and excerpt, so no further cut belongs inside the displayed option. Counts use the Qwen3-VL tokenizer on the original serialization, including attributes, indentation, and newlines. The figure abbreviates this text, so its displayed width does not represent token count. Figure 3a examines Qwen3-32B attention at layer 48 after 100 generated tokens in a separate WebVoyager prompt. Blue cells mark each head's top 10% of tokens over the full context. An offline structural partition retains the complete 20 token element 1790, including its `div` marker, attributes, and child text "Sports." The fixed boundary at token 19,328 splits the element before its tag ending and child text. Appendix A.7 retains the full diagnostic.

Context composition. We tokenize saved messages and parsed responses from 417 recorded steps across 44 WebVoyager tasks (He et al., 2024) using the Qwen3-VL tokenizer (Bai et al., 2025). The current DOM occupies a median of 22.5% of this text, with an interquartile range of 13.1% to 37.6%. Across 20 complete trajectories with at least five steps, the DOM share varies by more than ten percentage points in each, with a median range of 30 points. Across all 20 steps of ESPN–33, the DOM occupies 30.5% to 45.0% of the recorded text. History grows from 26 to 2,619 tokens, raising its share from 0.2% to 20.1%. The recorded context composition combines history and state into one role, while the history counts above exclude state. The proportions include the completed response but exclude chat template and tool schema overhead, so they do not measure full KV occupancy.

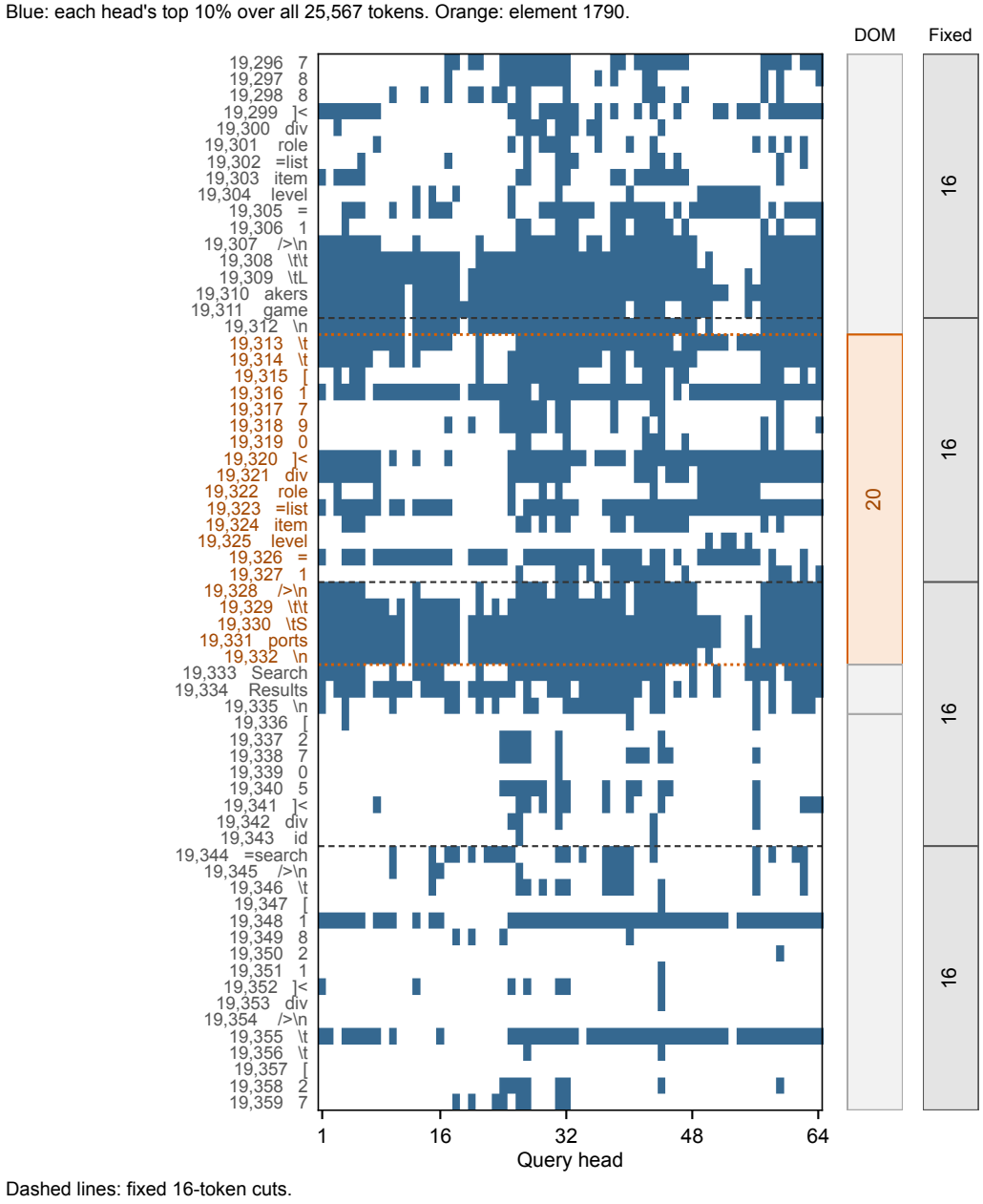

Figure 8: **Attention structure need not align with fixed-size blocks.** Recorded diagnostic from a Qwen3-32B replay: prompt 0, decode step 100, layer 48. Rows show 64 consecutive decoded tokens and columns show all 64 query heads. Blue indicates per-head top-10% attention-rank membership over the full context, not raw attention magnitudes. Orange bounds the complete 20-token element 1790; black dashed lines mark fixed 16-token cuts. Side strips show the corresponding semantic chunks and fixed blocks. This single case illustrates boundary mismatch, not measured selector accuracy or a performance improvement.

A.7 Extended attention boundary diagnostic

Attention structure and positional boundaries. Figure 8 illustrates why fixed token counts can be an awkward selection granularity for web observations. The heatmap marks tokens in each

head's top 10% by attention weight in one captured decoding step; it is an attention-rank diagnostic, not the output of a sparse selector. The highlighted subtree is a real serialized `div` element with identifier 1790, attributes `role=listitem level=1`, and the text child "Sports." Its marker is at indentation depth 2 and its text is at depth 3. The following line, "Search Results," returns to depth 0, establishing the subtree's end. The unchanged offline structural chunker with a 32 token cap emits this entire element as chunk 728, covering tokens 19,313–19,332, including whitespace. The fixed boundary at 19,328 falls before the tag ending `/>`, splitting the element into fragments of 15 and 5 tokens. The two overlapping blocks cover tokens 19,312–19,343, so retaining both includes 12 positions outside the element. This example motivates selection units that keep an element's identifier, attributes, and text together. Attention still varies across heads and within the chunk: the diagnostic motivates BrowSA's granularity choice but does not establish a task-success or latency gain.

Source and extraction. We use the recorded Qwen3-32B attention at layer index 48 after 100 generated tokens, with 25,467 prefill tokens and 25,567 context tokens in total. For each of 64 query heads, we mark the 2,556 largest stored attention weights over the full context before cropping. The main figure shows 32 consecutive tokens and two complete fixed blocks; the appendix adds 16 tokens on each side. We selected this illustration by inspecting complete DOM subtrees across three recorded prompts, four layers, and three decoding steps, seeking a clear attention concentration across a structural boundary split by fixed blocks. The rank threshold and all 64 query heads are unchanged, and the example does not estimate the prevalence of this pattern. The partition is an offline replay of `chunk_v2_prototype.py`, not a selection recorded by the serving system. The bundled plotting data record the input hashes, pinned tokenizer, original token IDs, and decoded pieces, and include an extraction script that verifies the complete subtree against the original prompt.

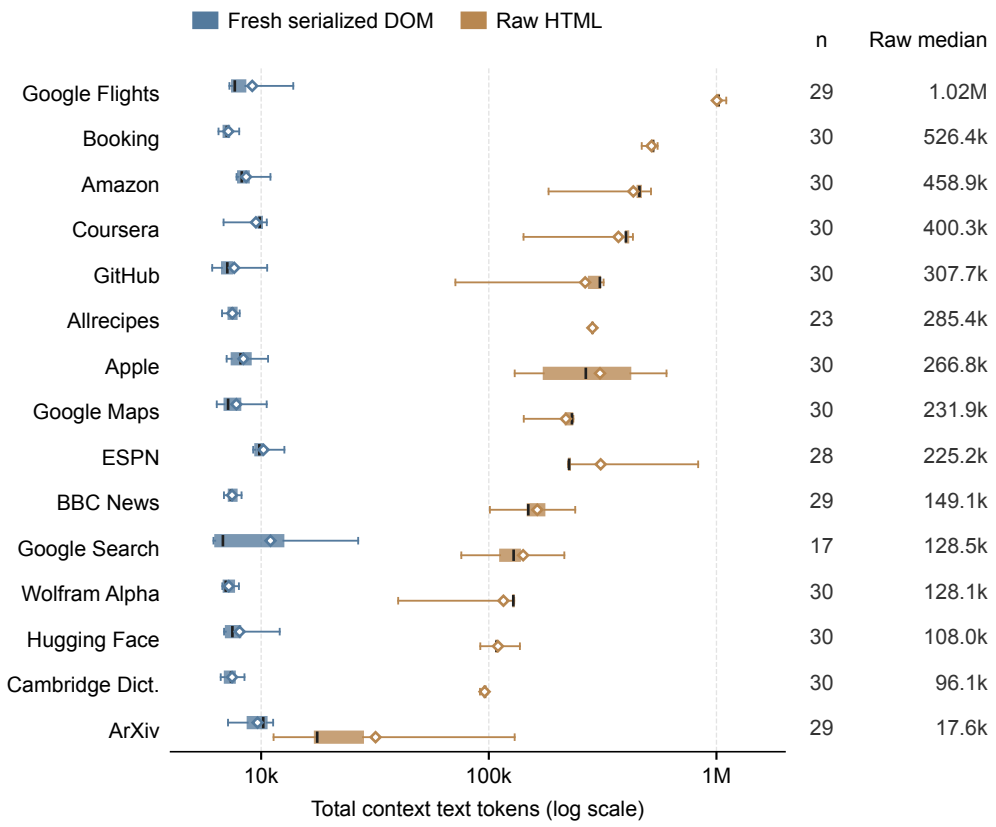

Figure 9: **Context length with raw HTML.** Paired URL revisits replace the page representation while retaining the recorded nonpage context. Boxes span the interquartile range with median lines, whiskers the 5th to 95th percentiles, and diamonds mark the mean. The right columns give retained captures out of 30 sampled steps per website and the median raw HTML context length. Both variants use Qwen3-32B text tokens.

A.8 Context length with raw HTML

Figure 9 measures how much context is required when the webpage observation contains raw HTML. For each sampled step, we keep the recorded system prompt, task, history, and other state text fixed and insert either freshly captured HTML or browser-use's serialized DOM. These paired measurements compare the two representations for the same revisited page. They do not recover the original HTML behind Figure 1.

Sampling and page capture. We use an independent local browser-use trajectory collection for all 643 WebVoyager tasks (He et al., 2024), containing 5,871 recorded input contexts. The records retain model messages, screenshots, and URLs, but not raw HTML. We exclude nine steps without an unambiguous current webpage URL and four PDF observations, then uniformly sample 30 steps without replacement from each task website. The fixed sample contains 450 steps from 323 tasks. For 66 sampled steps, we recover browser-use's shortened URL from the saved full URLs by verifying its hash. On September 22, 2026, we revisit each URL in an isolated headed Chrome 153 session with a 1280×720 viewport and browser-use 0.12.1. We wait at least seven seconds after navigation and allow up to 20 additional seconds for document loading before capture. Failed captures are retained in the audit and are not replaced by new samples.

Representations and counting. Raw HTML is the `outerHTML` of the main document's root element after rendering, including inline scripts, styles, and hidden markup. It excludes external resource bodies, child frame documents, and shadow tree interiors. The serialized DOM can also expose frame and shadow content, which is absent from this main document HTML. We retain browser-use's default 40,000 character limit for the serialized DOM and leave the HTML untruncated. Both page payloads are inserted verbatim, without another round of URL shortening. We retokenize each complete modified text part with the Qwen3-32B tokenizer and sum over the input messages. This preserves tokenization effects at the replacement boundary. Counts exclude screenshot vision tokens, chat template and tool schema overhead, and the step's generated response. This is a length measurement, and oversized contexts are not submitted to a model.

Measured lengths and coverage. We retain 425 paired measurements from 308 tasks and 115 distinct requested URLs. The other 25 samples comprise 15 HTTP errors, eight DOM extraction failures, one navigation failure, and one expired flight date error. Across task websites, median raw HTML context length ranges from 17,627 to 1,021,545 tokens. The median paired ratio of HTML context length to serialized DOM context length is $25.2\times$ across the retained steps. None of these DOM captures reaches the 40,000 character limit, so truncation does not drive the observed difference. Repeating the HTML capture after DOM extraction changes total context length by at most 0.96%.

Scope. Revisiting a URL does not restore historical cookies, form inputs, scroll positions, or other interaction state. The comparison therefore measures fresh page representations with recorded nonpage context, rather than an exact replay of the historical agent run. Website groups follow the task's assigned website, including any alternative pages visited by that task. The Allrecipes and Booking samples each revisit one requested URL, so recorded steps are not independent page samples. Sampling assigns equal numbers of steps to each website before capture failures, rather than reproducing the full benchmark's step distribution. The reported distributions are conditional on successful capture and inspection. We use the lengths to characterize representation cost, without inferring action quality or latency from them.

B Related Work

Efficient Web-Agent Inference. Web agents process webpage observations and interaction histories over multi-step trajectories. Benchmarks such as WebArena, Mind2Web, WebVoyager, and WorkArena/BrowserGym establish realistic settings for web interaction, while AgentOccam studies how observation and action representations affect agent behavior (Zhou et al., 2024; Deng et al., 2023; He et al., 2024; Drouin et al., 2024; Le Sellier de Chezelles et al., 2025; Yang et al., 2025b). Prior work compresses or restructures observations and histories through functional regions, DOM downsampling, pruning, contextualized representations, retrieval, selective refresh, and context compression (Kwon & Lee, 2026; Schiepanski & Piël, 2025; Zhang et al., 2026; Lee et al., 2025; Kerboua et al., 2025; Gaur & Lane, 2026; Kang et al., 2026). Aggressive observation reduction may remove useful information (Enomoto et al., 2026). Compression can also add latency through the extra compressor call (Kang et al., 2026). MarkupLM, DOM-LM, and WebFormer instead encode markup or DOM structure as representation biases (Li et al., 2022; Deng et al., 2022; Wang et al., 2022), while speculative decoding, Skim, RouteLLM, FrugalGPT, and model cascades reduce generation cost through drafting, fast paths, or model routing (Leviathan et al., 2023; Chen et al., 2023; Ong et al., 2025; Chen et al., 2024a; Wong et al., 2026). These directions modify the input or generation procedure. We study a complementary problem: retaining the full prompt while selecting its cached representations during decoding. A later query can revisit a chunk omitted by an earlier one. The DOM serves as a structural prior for this selection, without assuming that every subtree is a functional region.

Sparse Attention. Sparse attention reduces computation or KV access by selecting which parts of the context are attended. Classical methods impose local, global, random, hashed, or routed sparsity patterns (Child et al., 2019; Kitaev et al., 2020; Beltagy et al., 2020; Zaheer et al., 2020; Roy et al., 2021). For pretrained dense models, inference-time methods instead select relevant KV entries dynamically through query-aware retrieval, adaptive sparsity, sampling, or reuse across heads and layers (Tang et al., 2024; Ribar et al., 2024; Yang et al., 2024; Zhu et al., 2026; Chen et al., 2025b; Yang et al., 2025c; 2026), while related work targets sparse prefill or learns sparsity during training

or post-training (Jiang et al., 2024; Lai et al., 2025; Xu et al., 2025; Yuan et al., 2025; Lu et al., 2025; Gao et al., 2025; DeepSeek-AI, 2025). A separate design choice is the selection unit: prior methods form groups from latent representations, fixed token spans, or semantic boundaries (Hooper et al., 2025; Liu et al., 2025b;c; Zhu et al., 2025; Wu et al., 2026; Chen et al., 2025a; Li et al., 2026). SabreCoder combines semantic code chunks with different granularity for local code and retrieved snippets during prefill (Wang et al., 2026). BrowSA instead defines sparse-access units from DOM hierarchy and message structure in the web-agent prompt.

Variable-length units introduce additional challenges in both ranking and execution. Quest scores fixed pages using coordinate-wise key extrema (Tang et al., 2024). ArkVale studies alternative geometric summaries, LServe separates fine-grained logical scoring from physical pages, and EntmaxKV develops statistical page-level estimation for entmax attention (Chen et al., 2024b; Yang et al., 2025d; Duarte et al., 2026). For BrowSA, unequal semantic-chunk lengths make extrema-based ranking systematically length dependent, motivating an explicit calibration of the envelope-width term. Other methods adapt sparsity budgets across layers or heads (Cai et al., 2025; Feng et al., 2025; Wang et al., 2025), or vary block granularity (Chen et al., 2025a; Liu et al., 2026). The Sparse Frontier compares sparsification choices across tasks and inference settings (Nawrot et al., 2026), while Joshi et al. examine the opportunities and limitations of inference-time context sparsity (Joshi et al., 2026b). Finally, semantic units need not align with physical pages, so finer logical selection does not necessarily yield lower physical KV traffic. BrowSA therefore treats semantic selection and page locality as distinct problems, linking its sparse-attention design to the serving mechanisms.

LLM Serving Systems. KV-cache systems reduce serving cost through retention, memory management, and optimized execution. StreamingLLM, H_2O, Scissorhands, FastGen, and SnapKV reduce the resident KV set using sink/window, importance, or prompt-dependent policies, while ArkVale supports recall of evicted pages (Xiao et al., 2024; Zhang et al., 2023; Liu et al., 2023; Ge et al., 2024; Li et al., 2024; Chen et al., 2024b). PagedAttention, TensorRT-LLM, QServe, vAttention, and FlashInfer provide efficient abstractions and kernels for KV allocation and execution (Kwon et al., 2023; NVIDIA, 2023; Lin et al., 2025; Prabhu et al., 2025; Ye et al., 2025). LServe and Vortex extend this systems support to sparse attention (Yang et al., 2025d; Chen et al., 2026b), while RetrievalAttention, InfiniGen, PQCache, ShadowKV, and RetroInfer selectively access KV across memory tiers (Liu et al., 2025a; Lee et al., 2024; Zhang et al., 2025; Sun et al., 2025; Chen et al., 2026a).

DynSplit-KV and AB-Sparse study variable-size selection units and their efficient execution (Ye et al., 2026; Liu et al., 2026), while PRKV restructures page-level retrieval for offloaded KV (Wu et al., 2025). Context-reuse systems separately exploit repeated context across requests (Zheng et al., 2024; Yao et al., 2025; Hu et al., 2025; Pan et al., 2025). These systems establish the substrate for separating logical access from physical KV layout. BrowSA builds on this separation: semantic chunks remain the selection units, while a query-independent ordering derived from their key summaries is used only for physical placement.