SALT: Salience-Aware Lexical Trie for Long-Context Compression

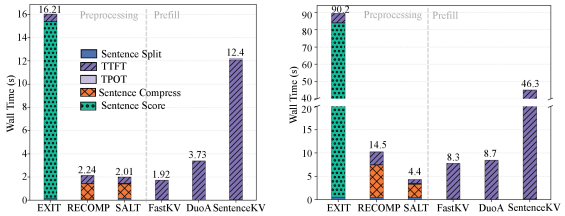
Anonymous ACL submission

Abstract

As large language models (LLMs) process increasingly longer prompts, computation and KV-cache memory costs have emerged as major bottlenecks in inference systems. Existing input-level prompt compression methods address this, but rank each sentence by a scalar relevance score, treating the document as an unstructured pool of words and sentences. Under tight budgets, this causes *theme collapse*, where the dominant theme(s) of a document consumes the budget, discarding less-frequent yet task-relevant themes. Preserving thematic coverage instead requires allocating the budget across recurring themes rather than scoring sentences in isolation. To this end, we propose SALT, a model-agnostic extractive framework that organizes per-sentence keywords into a trie ordered by sentence frequency (SF), a lightweight, reusable proxy for document thematic structure. This trie-based organization smooths memory allocation and prevents dominant themes from monopolizing the budget. Multi-anchor retrieval activates trie nodes labeled by query keywords at any depth, and the trie persists across dialogue turns, supporting multi-turn use without re-encoding the document. By preserving document themes, SALT reduces the prefill computation and memory cost of long-context prompts while remaining composable with KV-cache methods that target decoding-time latency and memory.

1 Introduction

Large language models (LLMs) have become essential for a wide range of natural language tasks, and their utility increasingly depends on their ability to process long-context inputs containing tens of thousands of tokens. These long contexts are central to document summarization, multi-hop question answering, and retrieval-augmented generation (RAG), where relevant information often spans long documents or multiple sources. To meet this need, recent open and proprietary models have expanded their context windows beyond a hundred thousand tokens (Dubey et al., 2024). This expansion has unlocked new capabilities, but it has also exacerbated the computational and memory costs of long-context inference. Each additional input token increases prefill computation, expands the key-value (KV) cache held in GPU memory throughout inference, and spreads attention across more positions (Dao, 2024). As a result, latency, memory usage, and generation accuracy become increasingly constrained by input length, making prompt compression or reduction before inference one of the most direct ways to improve long-context efficiency without modifying the underlying model.



(a) 32k context length. (b) 128k context length.

Figure 1: Wall-time for preprocessing methods (EXIT, RECOMP, SALT) and prefill/KV cache methods (FastKV, DuoA, SentenceKV), averaged over 30 inputs at 32k and 128k context lengths. Bars show TTFT breakdown and averaged 64-token decoding as TPOT. All methods run with 20% of context length of the input.

Existing methods address the latency and memory bottlenecks of long-context inference at two main points in the pipeline. Some methods reduce inference cost within the model through KV cache reduction, token eviction, sparse attention, or semantic cache management (Zhang et al., 2023; Xiao et al., 2025; Yan et al., 2026). These techniques modify attention computation or cache management during inference, reducing decoding-time latency and memory footprint. However, they require integration into the LLM inference stack, and

the full prompt must still be processed during prefill before any internal reduction takes effect. Other methods instead compress the input prompt before inference (Xu et al., 2024; Hwang et al., 2025; Liskavets et al., 2025; Zhang et al., 2025; Jiang et al., 2023; Pan et al., 2024). Because these methods shorten the text before it enters the target model, they can lower prefill computation, the KV cache size, and Time-To-First-Token (TTFT), while remaining complementary to KV cache methods that optimize later stages of inference. Figure 1 illustrates this pipeline-level distinction through a wall-time comparison of recent preprocessing and KV-cache methods at 32k and 128k context lengths.

Despite these advantages, preprocessing still leaves a distinct question of what structure the compressed prompt should adopt. Many extractive and pruning-based preprocessing methods operationalize this question by assigning scalar utility to local units, using retrievers, classifiers, contrastive encoders, proxy language models, attention signals, or query similarity, and then retaining high-utility units under a fixed budget (Xu et al., 2024; Hwang et al., 2025; Liskavets et al., 2025; Zhang et al., 2025; Jiang et al., 2023; Pan et al., 2024). Even when the utility is context-aware, the final decision often remains a one-dimensional ranking over candidates, further discussed in Appendix A. This abstraction is efficient, but it leaves theme coverage implicit rather than allocating the budget across the recurring themes of the document. Classical retrieval and summarization work makes this coverage concern explicit by balancing relevance with novelty, diversity, or representativeness (Carbonell and Goldstein, 1998a). Under tight budgets, scalar ranking can therefore overrepresent the dominant theme and omit a rarer theme. In multi-hop QA (Yang et al., 2018; Trivedi et al., 2022), for instance, it may retain several passages about the main entity while dropping the bridge sentence that connects it to the second entity. We refer to this coverage failure as *theme collapse*.

To address this gap, we propose SALT, a model-agnostic extractive framework for preprocessing long-context inputs before inference. Rather than letting a single global ranking determine which sentences enter the compressed prompt, SALT first constructs a lightweight representation of the document's thematic structure and uses it to allocate the compression budget across recurring themes. Sentences are anchored by extracted keywords, and the frequency of these keywords across the document provides a document-derived salience signal rather than one learned from a task-specific ranker. The selected sentences are then reconstructed in document order as a plain-text prompt, making SALT usable with any downstream language model and complementary to architectural methods that target memory use or decoding latency. Our main contributions are:

- We identify theme collapse, the loss of minor themes when scalar-ranked compressors are pushed to tight budgets, and reformulate extractive compression to allocate budget across document-derived lexical themes before sentence selection rather than as a post-hoc diversity correction.
- We propose SALT, a model-agnostic method for sentence-level prompt compression. SALT relies on a lightweight encoder to extract per-sentence keywords, organizes the salient ones into a trie ordered by sentence frequency, and allocates budget across trie branches before sentence selection.
- SALT outperforms prior preprocessing methods across accuracy, latency, and memory. It also is on par with state-of-the-art KV-cache techniques on latency and memory usage while remaining usable across NVIDIA GPUs generations.

2 Related Work

Prior work has studied ways to reduce the cost of long-context inference through prompt reduction, cache compression, or attention-side optimization. We discuss these directions in this section.

Input-level prompt compression shortens the textual input before it reaches the target LLM. Sentence-level methods such as RECOMP, EXIT, CPC, and Sentinel select or rewrite textual units using learned retrievers and proxy-model signals (Xu et al., 2024; Hwang et al., 2025; Liskavets et al., 2025; Zhang et al., 2025). Token-level compressors such as LLMLingua and LLMLingua-2 remove less informative tokens to achieve stronger compression ratios (Jiang et al., 2023; Pan et al., 2024). These methods are closest to SALT in the inference pipeline because they reduce the number of tokens processed during prefill. However, their selection abstraction has limitations: most reduce compression to scoring independent units and filling a budget with high-scoring candidates. SALT instead builds a document-level lexical theme struc-

ture and allocates budget across theme branches before sentence selection.

Another line of work addresses long-context input inside the target model by compressing KV states and pruning attention or tokens to reduce memory or latency (Zhang et al., 2023; Li et al., 2024; Xiao et al., 2025; Dongwon Jo, 2025; Zhu et al., 2025). These approaches share SALT's efficiency goal but operate at a different stage of the inference stack. They require access to model internals or cache management, and the original prompt is still processed by the target model before or during internal compression. SALT is complementary: it outputs a shorter plain-text prompt before prefill, remains model-agnostic, and can be composed with cache-side methods.

SALT is also related to classical extractive summarization and diversity-aware retrieval. Maximal Marginal Relevance (MMR) balances query relevance against pairwise novelty, while submodular summarization optimizes coverage and diversity under a budget (Carbonell and Goldstein, 1998b; Lin and Bilmes, 2011). These objectives show that compression should not greedily select only the highest-scoring units under an independent per-unit score. SALT differs in where diversity enters the decision. MMR and submodular objectives typically add novelty or set coverage during candidate selection after units have been scored. SALT makes coverage the primary allocation objective: budget is first distributed over document-induced lexical theme branches, and sentences are selected within those branches afterward. Thus, SALT performs pre-prefill extractive compression whose selection unit is the sentence but whose allocation unit is the lexical theme branch. This distinction is important under tight compression budgets, where late-stage diversity penalties cannot recover branches that never receive budget in the first place.

3 Method

SALT compresses a document into a sentence-level subset of bounded size while preserving coverage of its recurring lexical themes. We use lexical themes rather than semantic themes because compression decisions must remain lightweight, reusable across turns, and stable under small embedding perturbations. Lexical recurrence provides a sparse and interpretable approximation of document thematic structure while avoiding repeated pairwise semantic comparisons during traversal.

We organize SALT into two phases. An *indexing* phase estimates these themes from sentence-level keyword statistics and organizes the document into a salience-aware lexical trie. A *selection* phase chooses a sentence subset by traversing this structure under a target word budget, either unconditionally (*summary* mode) or with a query-dependent bias (*query* mode). These phases are separated for both logical and practical reasons. Logically, theme estimation and budget-constrained selection are distinct operations: the former defines the document structure, and the latter decides which parts of that structure can be represented under the budget. Practically, in multi-turn settings, such as a conversational agent answering successive questions over a long document, the same indexed structure can be queried repeatedly by turns whose keywords need without overlap. Designing the indexing artifact for repeated, query-conditioned access amortizes the encoding cost across turns and gives selection a uniform interface across modes. Figure 2 gives the overview of SALT.

3.1 Indexing

Given a document D partitioned into N sentences, we construct a candidate keyword ranking for each sentence using the lightweight open-source encoder BGE-small-en-v1.5 (Xiao et al., 2023). Specifically, we use [CLS] attention (Ding and Luo, 2021; Devlin et al., 2019) to rank content words within each sentence. This attention ranking is used only as a proposal signal, not as a faithful estimate of word importance; the number of words retained is determined by the reconstruction criterion described. Sentences are encoded within a 512-token window containing its surrounding context, while keyword selection is applied to the words of the target sentence. The full encoding protocol appears in Appendix B.

Within each sentence, we rank content words by descending [CLS] attention and incrementally form prefixes of this ranking. At step t, we compute the cosine similarity c_t between the mean embedding of the top-t words and the mean embedding of the full sentence. Although c_t tends to increase as more words are added, it is not guaranteed to be monotone: a newly added word can move the subset mean away from the full-sentence mean and produce a local dip. We therefore apply knee detection (Satopaa et al., 2011) to the monotone envelope

$$\bar{c}_t = \max_{\tau \le t} c_\tau, \qquad t = 1, \ldots, n_i, \tag{1}$$

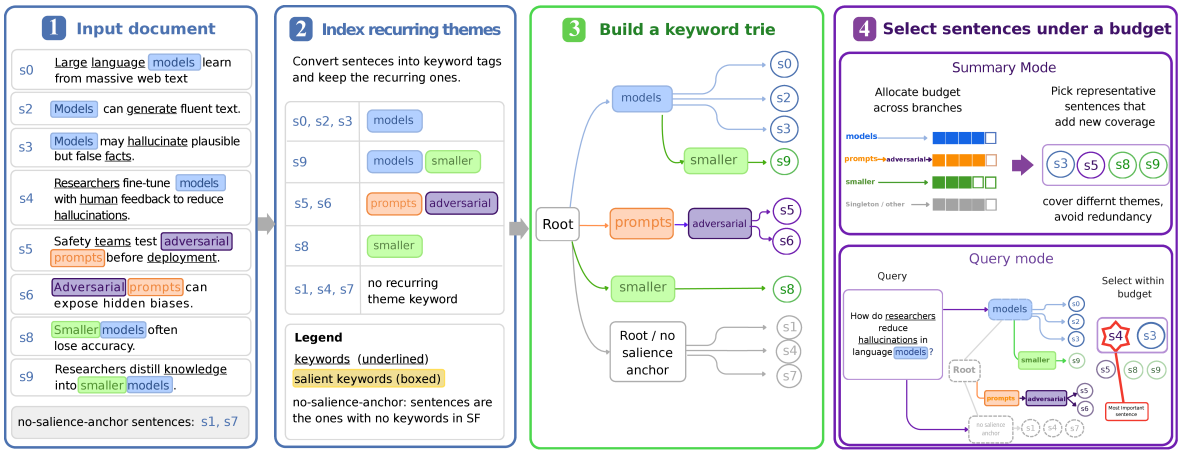


Figure 2: Overview of the SALT pipeline.

where n_i is the number of content words in sentence i. The running maximum suppresses local dips and yields a non-decreasing reconstruction curve on which the kneedle criterion is defined. The resulting knee k_i determines how many words are retained, but we cap it at 40% of n_i to prevent diffuse sentences from promoting most of their words into the keyword set. The top-k_i content words in the attention ranking form K_i; all lower-ranked words are excluded from the keyword index.

Over the per-sentence keyword sets $\{K_i\}_{i=1}^{N}$, we compute the sentence frequency (SF) as

$$\mathrm{SF}(w) = \big|\{i : w \in K_i\}\big|, \tag{2}$$

which measures how often a keyword appears as a selected anchor across sentences. A high sentence frequency indicates that the keyword participates in the document's recurring lexical structure, rather than appearing only in a local context. The salience set $\mathcal{S}$ retains keywords whose SFs are above the p-th quantile of this distribution (default $p = 0.9$), as shown in Figure 2, panel 2. This pruning removes low-frequency anchors and bounds the size of the trie, but it does not determine which themes are ultimately represented; coverage is imposed later by budget allocation across trie branches. Importantly, pruning affects only the indexing and scoring vocabulary. Since selection operates on whole sentences, any non-indexed word in a selected sentence remains in the compressed output. For selection, each retained keyword is assigned the normalized salience weight

$$\widehat{\mathrm{SF}}(w) = \frac{\mathrm{SF}(w)}{\mathrm{SF}_{\max}},$$

where $\mathrm{SF}_{\max} = \max_{u \in \mathcal{S}} \mathrm{SF}(u)$.

3.2 The Keyword Trie

The salience set $\mathcal{S}$ and the per-sentence keyword sets define a reusable lexical representation of the document. We organize this representation as a *keyword trie* $\mathcal{T}$ whose internal nodes are labeled by salience-set keywords and whose leaves store sentence identifiers. For each sentence s_i, we form $T_i = K_i \cap \mathcal{S}$, sort T_i by decreasing sentence frequency, and insert the resulting sequence as a root-to-leaf path ending at s_i. Sentences with common highest-salience anchors will share a trie prefix, while branch points record where their secondary anchors diverge. This structure preserves keyword co-occurrences that would be lost by assigning each sentence only to its highest-frequency keyword.

Coverage is measured over keyword mass in a subtree rather than over the number of sentences selected from it. For a trie node v, let $\mathcal{D}(v)$ denote its descendant sentences and let

$$\Gamma(v) = \bigcup_{s_i \in \mathcal{D}(v)} T_i$$

be the keyword signature of its subtree. Given a partial output $\mathcal{R}$, the keywords covered at v are

$$C_v(\mathcal{R}) = \Gamma(v) \cap \bigcup_{s_i \in \mathcal{R} \cap \mathcal{D}(v)} T_i.$$

The uncovered mass of v is then

$$U_v(\mathcal{R}) = \sum_{w \in \Gamma(v) \setminus C_v(\mathcal{R})} \widehat{\mathrm{SF}}(w). \tag{3}$$

Thus, a branch can be represented by a small number of sentences when those sentences cover its high-salience keyword signature. Additional sentences that repeat already covered anchors contribute little new mass, while sentences containing uncovered anchors reduce $U_v(\mathcal{R})$.

The trie also supports query-conditioned access without restricting traversal to ordinary prefix descent. Because the same keyword may appear at different depths depending on which anchors outrank it in each sentence, a query keyword activates all trie nodes with the corresponding label. Selection then unions the descendant regions of the activated nodes and applies the same coverage principle within those regions. This *multi-anchor activation* allows query mode to recover sentences where a query-relevant keyword appears as either a primary anchor or a secondary co-occurrence.

SALT trie is reusable across budgets, modes, and turns. Since indexing is independent of any particular query, successive queries over the same document can traverse the trie while changing only the activated nodes and selection scores. This separation amortizes the encoding cost across repeated accesses and gives summary-mode and query-mode compression a common document representation.

3.3 Budget-Constrained Selection

Selection traverses the trie under a target word budget B and returns a sentence subset $\mathcal{R}$ in original document order, where $\mathcal{R}$ denotes the current partial output and is updated after each admitted sentence. For a sentence s_i considered inside branch b, SALT scores the sentence by the reduction it would produce in the branch's uncovered mass:

$$\Delta_b(s_i \mid \mathcal{R}) = U_b(\mathcal{R}) - U_b(\mathcal{R} \cup \{s_i\}). \quad (4)$$

The sentence score is calculated as

$$\text{score}_b(s_i \mid \mathcal{R}) = \Delta_b(s_i \mid \mathcal{R}) L(n_i) I(s_i), \quad (5)$$

where $L(n_i)$ favors moderate-length sentences and $I(s_i)$ encodes sentence-level priors such as position and, in query mode, lexical match. Because the gain is marginal, sentences that repeat already covered anchors lose value, while sentences that cover new branch anchors remain competitive. Algorithm 1 gives the unified procedure.

In summary mode, $\mathcal{S}_{\text{eff}} = \mathcal{S}$, selection starts from the root. SALT first allocates the residual budget $B_\star$ across active depth-1 branches, then selects sentences within each branch using Eq. 5. The branch quota is a fixed floor plus a residual share proportional to $(M_b + \epsilon)^\alpha$, where M_b is the branch's remaining uncovered mass and $0 < \alpha < 1$. The floor reserves capacity for low-mass branches, while the sublinear exponent compresses high-mass branches. This allocation counters theme collapse by reserving coverage across recurring themes before local sentence scores can commit the budget to a dominant branch. Any unused budget is assigned by a final *GlobalFill* pass (Algorithm 1).

Algorithm 1: SALT selection.

Input : Trie $\mathcal{T}$, keyword sets $\{K_i\}$, budget B, optional query x.
Output : Sentence subset $\mathcal{R}$ with $\text{cost}(\mathcal{R}) \leq B$.

$\mathcal{R} \leftarrow \emptyset$; $\mathcal{S}_{\text{eff}} \leftarrow \mathcal{S}$
if *x is given* **then**
  $K_q, \mathbf{e}_q \leftarrow \text{QueryIndex}(x)$
  $\mathcal{S}_{\text{eff}} \leftarrow \mathcal{S} \cup (K_q \cap \bigcup_i K_i)$
  $\mathcal{A} \leftarrow \text{QueryAnchors}(\mathcal{T}, K_q, \{K_i\})$
  $\mathcal{R} \leftarrow \text{AnchorPhase}(\mathcal{A}, K_q, \mathbf{e}_q, \beta_q B)$
end
$B_\star \leftarrow B - \text{cost}(\mathcal{R})$
$\mathcal{B} \leftarrow \text{ActiveBranches}(\mathcal{T}, \mathcal{R})$
$\{\beta_b\}_{b \in \mathcal{B}} \leftarrow \text{BranchAllocate}(\mathcal{B}, B_\star)$
$\mathcal{R} \leftarrow \text{BranchPhase}(\mathcal{T}, \mathcal{R}, \{\beta_b\})$
$\mathcal{R} \leftarrow \text{GlobalFill}(\mathcal{T}, \mathcal{R}, B)$
return $\mathcal{R}$ sorted in document order

In query mode, SALT extracts query keywords K_q and the embedding $\mathbf{e}_q$. Query keywords already in $\mathcal{S}$ activate matching trie nodes; query keywords pruned from $\mathcal{S}$ but present in some K_i are reactivated through the stored sentence-keyword index. The effective salience set is therefore $\mathcal{S}_{\text{eff}} = \mathcal{S} \cup (K_q \cap \bigcup_i K_i)$, and query keywords receive larger mass when computing U_b. The anchor phase ranks activated candidates based on lexical-anchor overlap with K_q and positive embedding similarity to $\mathbf{e}_q$, admitting the top candidates and their immediate neighbors up to $\beta_q B$. The remaining budget is allocated using BranchAllocate and GlobalFill procedures as summary mode. As a result, β_q controls the relevance–coverage trade-off: smaller values favor summary-style compression, while larger values prioritize query-aligned evidence.

4 Experimental Evaluation

Datasets and Models We evaluate accuracy on the English subset of LongBench (Bai et al., 2023), covering six task categories: Single-Doc QA, Multi-Doc QA, Summarization, Few-Shot Learning, Synthetic, and Code, as well as QuALITY (Pang et al., 2022) and RULER (Hsieh et al., 2024) to assess long-context reasoning and retrieval. All methods use identical prompts and evaluation met-

Table 1: LongBench results on LLaMA-3.1-8B-Instruct (20% KV cache retention).

Method	Single-Doc QA	Multi-Doc QA	Summarization	Few-Shot	Synthetic	Code	Avg.
Llama-3.1-8B-Instruct							
Full-context	43.58	44.65	29.22	69.48	54.21	60.01	50.19
KV Cache Methods (20%)							
SnapKV	43.29	43.92	26.59	67.95	53.75	58.74	49.04
FastKV	43.31	44.10	26.61	68.36	53.72	59.26	49.23
SentenceKV	39.25	43.82	28.18	69.26	53.24	47.33	46.85
DuoAttention	34.71	36.67	24.30	58.02	50.81	54.33	43.14
Preprocessing Methods (20%)							
EXIT	31.50	23.77	24.94	58.89	13.5	37.45	31.68
RECOMP	35.88	41.09	24.16	52.06	50.77	37.37	40.22
CPC	38.91	39.42	24.97	51.67	51.50	21.84	38.05
Sentinel	39.85	41.66	26.15	38.78	51.55	38.83	39.47
SALT	40.05	41.42	26.95	62.21	53.37	37.06	43.51

rics following lm-eval-harness (Gao et al., 2024), ensuring differences are attributable solely to the compression method. All evaluated datasets include query except for the summarization section of the LongBench. For latency and memory profiling, we construct long-context inputs from the PG19 dataset (Rae et al., 2020). We present all main results on Llama-3.1-8B-Instruct (Llama Team, 2024), with additional results on Ministral-8B-Instruct-2410 (AI, 2024) included in the appendix.

Hardware We conduct most experiments on a compute node equipped with an NVIDIA H100 GPU and AMD EPYC 7R13 processor, except those in Section 4.3, where we additionally benchmark on NVIDIA V100, A100, and B200 GPUs to assess performance across hardware generations.

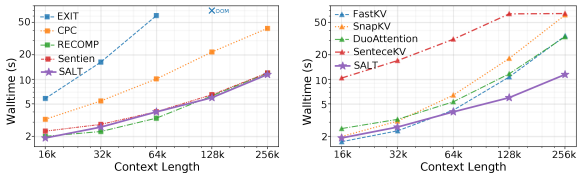
Configurations For comparative methods, we follow their published configurations with minimal changes for fair comparison. SnapKV runs at its reported defaults under a 20% budget. FastKV uses a 60% prefill and 20% decode budget, matching its paper. DuoAttention is run at 20% rather than its paper's best 50% setting to keep methods budget-matched. SentenceKV uses the original implementation unchanged. H2O uses chunked prefill at 8k. All KV-cache methods except H2O use FlashAttention2 (Dao, 2024). EXIT splits documents into sentences with spaCy (Honnibal et al., 2020) and scores each with the LoRA-tuned Gemma-2B classifier at threshold 0.5. RECOMP is used unchanged. CPC uses the released pretrained LoRA with a local Llama-3.1-8B-Instruct answer generator. Sentinel matches the paper's Qwen-2.5-0.5B proxy and trained detector. Minor preprocessing adjustments were made to H2O and Sentinel to prevent OOM on long inputs; details are provided in Appendix C.

4.1 Accuracy

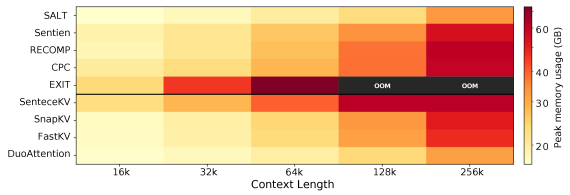
We report LongBench accuracy at a 20% memory budget across all methods, the regime where compression is most aggressive and methodological differences are most informative. The current state-of-the-art KV-cache methods SnapKV and FastKV recover within roughly one point of the full-context baseline, while SALT provides similar accuracy for most subcategories, with code degrading the overall score the most. The gap is structural. KV-cache methods prune after the model has read the full prompt and use the model's own attention as a salience signal at token granularity, whereas preprocessing methods must commit up front using an external relevance estimate. Within the preprocessing methods, SALT leads on average and wins across most categories for the 20% budget, showing it preserves the context and important sections of the input that causes other preprocessing methods to break. All preprocessing methods underperform on Code, where several KV-cache methods match full context, because code is line and token sensitive while preprocessing methods operate at sentence or paragraph level. Notably, SentenceKV, the one KV-cache method aggregating at sentence granularity, shows the same Code weakness, supporting a granularity explanation rather than a preprocessing versus KV-cache one. More results in Appendix C.

4.2 End-to-End Efficiency at Scale

Reducing the prompt only pays off if the reduction itself is cheap enough to run at scale. Preprocessing methods that score every chunk with an aux-



(a) Preprocessing (b) KV-cache



(c) Peak GPU memory across methods and context lengths.

Figure 3: End-to-end efficiency of SALT against preprocessing and KV-cache baselines on Llama-3.1-8B across context lengths from 16k to 256k tokens. Memory and Walltime are in log scale.

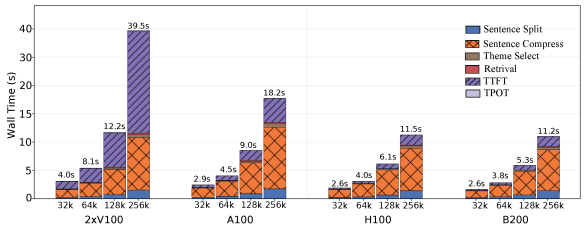


Figure 4: End-to-end latency and TPOT of SALT on Llama-3.1-8B-Instruct across different NVIDIA GPUs and raw-context lengths at a 20% retention budget.

Table 2: Peak GPU memory (GB) of SALT across prompt lengths, and NVIDIA GPUs architectures.

	Context length (tokens)				Scaling
GPU	32k	64k	128k	256k	256k / 32k
V100x2	20.10	22.50	27.87	40.22	2.00×
A100	18.30	20.65	24.14	33.29	1.81×
H100	17.90	20.20	23.30	32.80	1.83×
B200	16.41	17.86	20.67	25.72	1.57×

iliary model introduce overhead that grows with the input, while KV-cache methods inherit the cost of attention over the full prompt before they can compress it. We benchmarked SALT against four preprocessing baselines (EXIT, CPC, RECOMP, Sentinel) and four KV-cache baselines (FastKV, SnapKV, DuoAttention, SentenceKV) on Llama-3.1-8B at context lengths from 16k to 256k tokens, measuring peak GPU memory and walltime that includes a 128-token decode per prompt so that the cost of generation is also captured. Detailed values are reported in Appendix D.

Figures 3a and 3b show the latency picture. Among preprocessing methods, those that delegate selection to an auxiliary model (e.g., EXIT, CPC) inherit that model's forward pass at every input and degrade quickly as context grows. KV-cache methods generally amortize their selection at prefill and add little additional cost during decode; SentenceKV is the exception, as it performs sentence-level selection at prefill and continues to act through decode, paying extra cost on every generated token. Figure 3c shows a similar separation on memory, and across both axes SALT remains among the cheapest methods at every length tested.

4.3 Hardware Portability Across Context Lengths

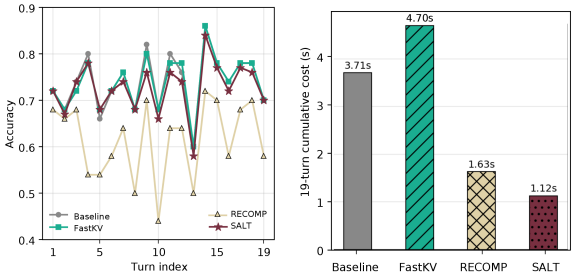
Most reduction methods depend heavily on newer GPU architectures to manage memory during the prefill phase. Systems like SnapKV fundamentally require custom Triton kernels and FlashAttention compatibility, strictly mandating modern accelerators to execute efficiently. Conversely, input level preprocessing methods bypass these limits but often introduce new dependencies by relying on external proxy models to run their selection algorithms.

To demonstrate how our approach avoids these bottlenecks, we evaluated SALT using an 8B backbone at a fixed 20% retention budget on PG19 datasets with contexts scaling up to 256k tokens. We compared four distinct GPU generations: a legacy Volta setup using two V100 units (32GB each) from 2017 against modern Ampere A100, Hopper H100, and Blackwell B200 accelerators.

Figure 4 shows that the older Volta hardware successfully processes 256k tokens. SALT is highly adaptable and does not depend on specific accelerator libraries. Furthermore, because SALT outputs a standard plain text prompt prior to prefill, it is not inherently restricted to NVIDIA GPUs, though we currently lack access to alternative architectures to present data on them. Detailed normalized walltime data is provided in Appendix E.

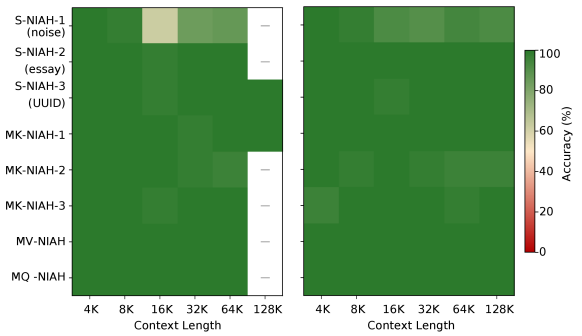
4.4 Per-Turn Cost in Extended Interactions

Single-turn benchmarks understate the compute footprint of compression in realistic deployment, where the same document is queried repeatedly across a conversation. We evaluate this regime on QuALITY with a 50-article subset and 972 question turns at a 20% budget, comparing an uncompressed baseline, FastKV, RECOMP, and SALT on Llama-3.1-8B-Instruct. The full setup and per-turn timing breakdown are in Appendix D. Fig-



(a) Accuracy (b) Cost

Figure 5: QuALITY at 20% budget, 50 articles, 972 turns. (a) Per-turn accuracy is bounded for all methods. (b) Cumulative compute over a 19-turn conversation diverges sharply.



(a) Baseline (b) SALT

Figure 6: Needle-in-a-Haystack (NIAH) results from RULER on Llama-3.1-8B-Instruct. The baseline uses the uncompressed prompt, while SALT retains 20% of the source context before inference. White cells indicate unavailable full-context runs whose final tokenized prompts exceed the model context window.

ure 5 shows that per-turn accuracy is essentially flat across 19 turns for every method, with SALT matching the baseline and FastKV throughout. Cumulative cost, however, diverges sharply and SALT achieves this accuracy within a fraction of FastKV and baseline.

4.5 Needle-in-a-Haystack (NIAH)

We evaluate SALT on the NIAH retrieval benchmark from RULER (Hsieh et al., 2024), which embeds a specific fact ("needle") within a long distractor context and queries the model to retrieve it. We report the eight NIAH variants spanning single-needle, multi-key, multi-value, and multi-query retrieval. As shown in Figure 6, SALT preserves NIAH accuracy across all context lengths matching the uncompressed baseline on NIAH tasks. As context grows, SALT's accuracy degrades more gracefully than the uncompressed model on harder variants, demonstrating that content-aware sentence selection retains the critical needle even under aggressive compression.

5 Conclusion

We introduced SALT, a lightweight, model-agnostic extractive framework that reframes prompt compression as preserving thematic coverage under a fixed word budget, using sentence frequency of lexical keywords as a reusable proxy. By organizing per-sentence keywords into an SF-ordered trie, SALT allocates budget across recurring themes before sentence selection and avoids the theme collapse caused by scalar ranking. The trie supports both summary-mode traversal and multi-anchor query retrieval, enabling multi-turn dialogue without re-encoding the document, while outputting plain text compatible with any downstream LLM and complementary to KV-cache optimizations. Across LongBench, QuALITY, and NIAH at a 20% budget, SALT leads preprocessing baselines on accuracy and matches or beats KV-cache methods on prefill latency and peak memory across context lengths.

Limitations

SALT has limitations on code-heavy tasks, where it underperforms both the full-context baseline and KV-cache methods that retain token-level granularity. Because SALT operates at the sentence level, it cannot reliably distinguish where one code unit ends and the next begins, often grouping or splitting code fragments in ways that break their semantics during retrieval. Lexical keyword extraction is also a poor fit for source code, where identifiers, operators, and structural tokens carry meaning that the [CLS] attention signal was never trained to capture. This is not unique to SALT as all preprocessing methods in our comparison degrade on Code for the same granularity and tokenization reasons.

SALT is also bounded below by its sentence-level granularity. Across our evaluations we observed that the task-relevant span in long-context datasets typically lies within 5–20% of the input and varies considerably across datasets, so we set 20% as a conservative lower bound. Token-level KV-cache methods do not face this floor and can compress more aggressively, but pay for it in attention-time memory and require full prefill before any reduction takes effect. SALT trades this finer reach for a model-agnostic, pre-prefill path that remains composable with those KV-cache methods downstream.

References

Mistral AI. 2024. Ministral-8b-instruct-2410. https://huggingface.co/mistralai/Ministral-8B-Instruct-2410. Large Language Model.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *Preprint*, arXiv:2308.14508.

Jaime Carbonell and Jade Goldstein. 1998a. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 335–336.

Jaime Carbonell and Jade Goldstein. 1998b. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*.

Thomas M. Cover and Peter E. Hart. 1967. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27.

Tri Dao. 2024. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Haoran Ding and Luo Luo. 2021. AttentionRank: Unsupervised keyphrase extraction using self and cross attention. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1919–1928.

Yulhwa Kim Jae-Joon Kim Dongwon Jo, Jiwon Song. 2025. Fastkv: Decoupling of context reduction and kv cache compression for prefill-decoding acceleration. *arXiv preprint arXiv:2502.01068*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and et al. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. The language model evaluation harness.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, Adriane Boyd, and 1 others. 2020. spacy: Industrial-strength natural language processing in python.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*.

Taeho Hwang, Sukmin Cho, Soyeong Jeong, Hoyun Song, SeungYoon Han, and Jong C. Park. 2025. EXIT: Context-aware extractive compression for enhancing retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 4895–4924, Vienna, Austria. Association for Computational Linguistics.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376, Singapore. Association for Computational Linguistics.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*.

Zehan Li, Xin Zhang, and ... 2023. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*.

Hui Lin and Jeff Bilmes. 2011. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*.

Barys Liskavets, Maxim Ushakov, Shuvendu Roy, Mark Klibanov, Ali Etemad, and Shane K. Luke. 2025. Prompt compression with context-aware sentence encoding for fast and improved llm inference. AAAI'25/IAAI'25/EAAI'25. AAAI Press.

Meta Llama Team. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. 2024. LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt

compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 963–981, Bangkok, Thailand. Association for Computational Linguistics.

Richard Yuanzhe Pang, Alicia Parrish, Nitish Joshi, Nikita Nangia, Jason Phang, Angelica Chen, Vishakh Padmakumar, Johnny Ma, Jana Thompson, He He, and 1 others. 2022. Quality: Question answering with long input texts, yes! In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5336–5358.

et al. Paszke, A. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, Chloe Hillier, and Timothy P Lillicrap. 2020. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*.

Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. 2011. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st International Conference on Distributed Computing Systems Workshops*, pages 166–171.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, junxian guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. 2025. Duoattention: Efficient long-context LLM inference with retrieval and streaming heads. In *The Thirteenth International Conference on Learning Representations*.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-pack: Packaged resources to advance general chinese embedding. *Preprint*, arXiv:2309.07597.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2024. RECOMP: Improving retrieval-augmented LMs with context compression and selective augmentation. In *The Twelfth International Conference on Learning Representations*.

Siyuan Yan, GUO-QING JIANG, Yuchen Zhang, Xiaoxing Ma, Ran Zhu, Chun Cao, and Jingwei Xu. 2026. Adamas: Hadamard sparse attention for efficient long-context inference.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Yong Zhang, Yanwen Huang, Ning Cheng, Yang Guo, Yun Zhu, Yanmeng Wang, Shaojun Wang, and Jing Xiao. 2025. Sentinel: Attention probing of proxy models for llm context compression with an understanding perspective. *arXiv preprint arXiv:2505.23277*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

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

A Dimensionality of Scalar Preprocessing

We now clarify the sense in which scalar preprocessing turns compression into a one-dimensional selection problem. The claim is not that existing compressors use simple models or ignore context. Many preprocessing methods use contextual encoders, retrievers, proxy language models, or query-aware signals. The limitation appears at the selection interface. Once each sentence, passage, or token is assigned a single utility value, the budget is usually filled by comparing candidates along that value, sometimes with a later diversity correction. This reduces a document with several recurring themes to a flat list of candidates.

A scalar list is efficient, but it does not specify how the budget should be distributed across the themes of the document. If many high-scoring candidates come from the dominant theme, top-ranked selection can spend most of the budget on that theme. Lower-frequency themes can then disappear, even when they are needed to represent the document or support downstream reasoning. We refer to this coverage failure as theme collapse.

Classical work in diversity-aware retrieval and summarization, including embedding-based selection over centroids or nearest neighbors, has long argued that relevance alone is not sufficient under a budget and that novelty, diversity, or coverage must enter the objective (Cover and Hart, 1967). SALT adopts this concern as an input-level compression principle. Instead of relying on a final ranked list, whether from a scalar utility model or from geometric proximity to a document centroid, to preserve coverage implicitly, SALT allocates budget across document-induced theme branches before selecting sentences within those branches.

To test whether scalar order alone is a reliable guide, we run a diagnostic summarization ablation at a 20% token budget. All compressed variants use the same embedding model, grouping procedure, and document-order reconstruction. The difference is how candidates are selected from the score distribution. As shown in Table 3, selecting only the top-ranked centroid candidates is not consistently superior. These results do not imply that every error comes from theme collapse. They show the narrower point needed here. A single global score is not a complete description of a candidate's value under a tight budget, because useful information can lie outside the top of the scalar ranking. This motivates explicit allocation across themes.

Table 3: Diagnostic ROUGE-L results at a 20% token budget. Full context is included only as an upper reference. All methods use document-order reconstruction.

Method	Gov.	QMS.	MNews	Avg.
Full context	35.14	25.78	26.73	29.22
Random	29.63	21.73	22.02	24.46
kNN	30.23	20.95	22.50	24.56
Centroid	29.64	20.45	21.62	23.90
Centroid + filt.	29.69	19.94	22.61	24.08
Lead + centroid	29.23	20.73	22.87	24.28
Spread selection	30.34	20.62	22.17	24.38
SALT (Trie)	32.77	23.99	24.08	26.95

SALT still uses sentence-level scores, but it does not let a single global ranking determine the compressed prompt. The trie first exposes recurring lexical branches in the document, and the retrieval procedure assigns budget across those branches. Sentence scores are then used within each branch. This separates the coverage decision from the local selection decision, reducing the chance that the dominant theme consumes the budget before minor themes can appear.

B Keyword Extraction via Transformer Attention

We extract per-sentence keywords from the input document by repurposing the internal attention patterns of a pretrained transformer encoder. Rather than using the model's output embeddings in the standard way, we hook into two intermediate signals: (i) the `[CLS]` token's attention weights from the final layer, which indicate per-token importance, and (ii) the per-token hidden-state vectors, which provide contextual embeddings for measuring how well a keyword subset reconstructs the full sentence meaning. The extracted keywords and their attention-derived importance weights constitute the atomic representational units for all downstream processing.

B.1 Model Selection

We employ `BAAI/bge-small-en-v1.5`, a 6-layer, 12-head BERT encoder with hidden dimension d=384. This model was selected for three architectural and empirical reasons.

First, its compact architecture (33M parameters, 6 layers) provides a favorable trade-off between representational capacity and inference cost. With documents routinely exceeding 10,000 tokens, the model must process dozens of packed chunks per document. A larger encoder would increase latency

without proportionate gains in attention quality for keyword extraction, which depends primarily on the final layer's attention distribution rather than deep semantic reasoning.

Second, BGE was trained via contrastive learning on sentence-level retrieval tasks, which directly optimizes the [CLS] token to aggregate discriminative sentence-level information. This training objective produces attention patterns where [CLS] selectively focuses on content-bearing tokens, precisely the signal we extract. Models trained with different objectives (e.g., masked language modeling alone) distribute [CLS] attention more uniformly, yielding less informative importance rankings.

Third, we evaluated BGE against GTE-small (Li et al., 2023) (identical BERT architecture, different training objective) on the same documents. Both models extract keywords via the same pipeline, but their attention distributions differ fundamentally. GTE's contrastive training with cosine similarity supervision produces more peaked attention, a small number of dominant topic terms receive disproportionate weight across most sentences. BGE's instruction-tuned retrieval training produces more distributed attention, assigning meaningful weight to a broader set of content words per sentence. In practice, GTE extracted $\sim$20% fewer unique keywords across the same document, with its top terms appearing at 2–3$\times$ the frequency of BGE's. For keyword-based downstream processing, this concentration is disadvantageous: when most sentences share the same few high-weight keywords, it becomes difficult to distinguish which sentences cover which specific aspects of the document. BGE's broader vocabulary provides the granularity needed to differentiate sentence-level content.

B.2 Dense Packing with Span-Local Renormalization

A standard approach processes each sentence through the encoder independently. For a document of N sentences averaging $\sim$28 tokens each, this requires N forward passes, each utilizing under 6% of the model's 512-token input capacity. Beyond the computational waste, isolated encoding deprives the model of cross-sentence context: the attention pattern for a sentence is computed without knowledge of what surrounds it, so the model cannot distinguish document-central terms from locally prominent but globally generic ones.

We address both limitations through *dense packing*: consecutive sentences are greedily concatenated into 512-token chunks, with a 2-sentence overlap between adjacent chunks to ensure boundary sentences receive context from both directions. This reduces the number of forward passes by approximately an order of magnitude while exposing each sentence to its neighborhood during attention computation.

The key challenge is recovering per-sentence keyword rankings from a chunk-level attention distribution. When multiple sentences share a single [CLS] attention vector, tokens compete globally. A keyword in one sentence may receive low attention simply because a different sentence in the same chunk contains higher-salience terms. We resolve this through *span-local renormalization*: for each sentence's token span $[s, e)$ within a chunk, the raw CLS attention scores are divided by their span sum:

$$\hat{a}_j^{(s_i)} = \frac{a_j}{\sum_{k=s}^{e-1} a_k}, \quad j \in [s, e) \tag{6}$$

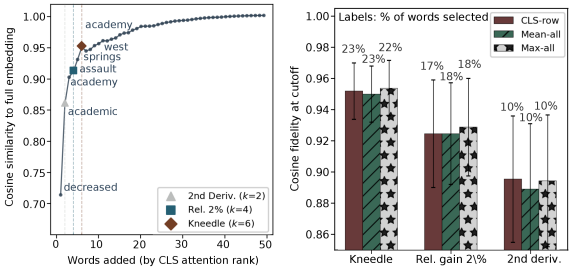
This produces a probability distribution that sums to 1 within each sentence. Critically, the global context that shaped the raw attention values is preserved, the model "saw" neighboring sentences when computing these scores, but the ranking is now relative to the sentence's own tokens. A term that the model considers important given the surrounding context will rank highly even if its raw score is modest compared to tokens in adjacent sentences.

For sentences appearing in two overlapping chunks, we retain the renormalized attention from the chunk where the sentence received the highest total raw attention mass (indicating the most informative context window), while raw attention scores are MAX-aggregated across chunks to preserve any importance signal observed in either context.

B.3 Kneedle-Based Keyword Selection

Given the per-word attention scores for a sentence, we must determine how many words qualify as keywords. A fixed threshold or fixed percentage would ignore the natural variation in how attention distributes across sentences of different lengths and information densities. Instead, we use a data-driven cutoff based on the geometry of an accumulation curve.

Words are ranked by descending attention and incrementally added to a subset. At each step t, we compute the cosine similarity c_t between the mean hidden-state embedding of the accumulated



(a) Cosine accumulation (b) Attention source × cutoff

Figure 7: (a) Cosine accumulation curve for a sample sentence: words added in CLS attention order progressively reconstruct the full-sentence embedding. The Kneedle cutoff (green) identifies the point of diminishing returns. (b) Comparison of three attention sources (CLS-row, mean-all, max-all) and three cutoff methods. Labels indicate the fraction of words selected. CLS-row with Kneedle achieves high fidelity at low selection rate.

subset and the mean embedding of the full sentence. This curve $\{(t, c_t)\}$ is monotonically increasing. It starts low when only one word is included and approaches 1 as the subset converges to the full sentence. The shape is characteristically concave: early words contribute large jumps in similarity (they carry disproportionate semantic weight), while later words contribute diminishing increments.

The Kneedle algorithm identifies the *knee* of this curve, the transition from steep to flat, by normalizing both axes to $[0, 1]$ and finding the point t^* that maximizes the perpendicular distance from the diagonal:

$$t^* = \arg \max_{t \leq \lfloor r_{\max} \cdot M_i \rfloor} \frac{|\tilde{c}_t - \tilde{t}|}{\sqrt{2}} \tag{7}$$

where $r_{\max} = 0.4$ caps the search at 40% of words to prevent over-selection. Content words (alphabetic, longer than 2 characters, non-stopword) ranked at or above t^* become the sentence's keyword set $\mathcal{K}_i$, each carrying its attention weight a_k.

We evaluated two alternative cutoff methods. The *relative gain drop* method (stop when the cosine gain at step t falls below 2% of the first step's gain) reacts to the initial steep drop and terminates prematurely, typically at $k{=}2$, selecting only $\sim 7\%$ of words, which is insufficient to capture the sentence's informational breadth. The *second-derivative* method (stop at the point of maximum deceleration) exhibits the same early-termination bias, since the largest deceleration in a concave curve always occurs at the transition from the first to second step. Kneedle, by contrast, detects the global shape transition rather than local rate changes, yielding a stable cutoff at $\sim 21\%$ of words with cosine fidelity of ~ 0.95 (Figure 7b).

We additionally compared CLS-row attention (our approach) against two alternative extraction signals: mean attention received by each token from all query positions, and maximum attention received from any single query position. CLS-row attention with Kneedle achieves the highest cosine fidelity at the lowest selection fraction, confirming that the `[CLS]` token's attention row is a more focused importance signal than aggregated all-token attention (Figure 7b).

B.4 Output Format

Phase 1 produces, for each sentence s_i: a keyword set $\mathcal{K}_i \subset \mathcal{V}$ with associated attention weights $\{a_k\}_{k \in \mathcal{K}_i}$, and the per-token hidden-state embeddings $\{\mathbf{h}_j\}$ from the final transformer layer. The keyword sets typically contain 5-7 content words per sentence (median 6 at average sentence length 28 words), with attention weights reflecting the model's assessment of each keyword's importance to the sentence-level representation.

C Accuracy

Method Configurations All methods are implemented in PyTorch (Paszke, 2019) 2.6.0 or 2.7.1, depending on each method's release requirements and compatibility with FlashAttention2. H2O uses chunked prefill at 8k to avoid OOM on long prompts. CPC replaces its LLMLingua GPT-3.5 evaluator with a local Llama-3.1-8B-Instruct pipeline so all methods share the same answer generator. Sentinel adds token-aware chunking at sentence boundaries to the preprocessing script, as the original implementation feeds the full context to the proxy in one shot and OOMs on long LongBench inputs.

C.1 Multi-Turn Evaluation on QuALITY

We evaluate on the QuALITY (Pang et al., 2022) dev split (v1.0.1, html-stripped): multiple-choice QA over long narrative documents, using a 50-article subset (972 question turns). Each question is issued as a single conversational turn and scored by `argmax` over the option-letter logits. All runs use LLAMA-3.1-8B-INSTRUCT in bf16 with SDPA attention on a single H100, with a 10-token decode budget for TPOT parity.

We compare four configurations at a 20% compression budget where applicable: an uncom-

Table 4: LongBench results on Ministral-8B-Instruct (20% KV cache retention).

Method	Single-Doc QA	Multi-Doc QA	Summarization	Few-Shot	Synthetic	Code	Avg.
Ministral-8B-Instruct							
Full-context	41.66	49.45	27.73	71	54.5	66.3	51.77
KV Cache Methods (20%)							
SnapKV	40.69	49.59	25.64	70.82	54.5	65.61	51.14
FastKV	41.07	49.38	25.33	70.75	54.75	65.49	51.13
DuoAttention	28.85	33.25	21.04	63.52	14.50	51.88	35.51
SentenceKV	35.41	26.73	20.29	54.14	43.50	48.29	38.06
Preprocessing Methods (20%)							
EXIT	32.66	32.28	23.97	65.37	13.25	40.12	34.61
RECOMP	35.11	50.10	23.24	54.71	52.50	40.90	42.76
CPC	39.05	43.99	24.17	52.17	50.25	11.49	36.85
Sentinel	39.72	42.29	25.34	61.88	50.25	43.64	43.85
SALT	39.87	45.51	25.36	66.59	52.25	42.91	45.42

Table 5: QuALITY @ 20% budget. 50 articles, 972 turns. Comp. = per-turn compression (ms). Σ_{19} = 19-turn cumulative cost (s).

Method	TTFT (ms)	Comp. (ms)	Acc.	HARD	Σ_{19} (s)
No compression	195	—	0.741	0.668	3.71
FastKV	138	—	0.739	0.664	2.62
RECOMP	53	46	0.615	0.543	1.88
SALT	38	11	0.726	0.657	0.91

pressed baseline; FastKV with 0.20 KV-cache retain rate; SALT with a per-article index built once and per-turn theme-conditioned retrieval; and RECOMP-extractive with the published `fangyuan/nq_extractive_compressor`, re-embedding sentences per query. RECOMP's encoder is run in bf16 under `torch.autocast` (embeddings cast to fp32 for scoring) for a 3× speedup with no measurable accuracy change. SALT and RECOMP are matched on input-token budget (∼1.1–1.2k tokens); FastKV leaves the prompt unchanged at ∼5.7k tokens.

At matched budget, SALT essentially preserves baseline accuracy (72.6% vs. 74.1% baseline, 73.9% FastKV; $\Delta < 1.5$ pp) and outperforms RECOMP-extractive by ∼11 points (72.6% vs. 61.5%). Per-turn cost differs sharply: RECOMP re-encodes the document on every query at 46 ms/turn, while SALT amortizes a 190 ms index across the conversation and pays only ∼11 ms/turn thereafter. Over a 19-turn dialogue, total compression+prefill cost is 3.71 s (baseline), 2.62 s (FastKV), 1.88 s (RECOMP), and 0.91 s (SALT), a 4× end-to-end speedup over the baseline at near-equal accuracy, and a 2× speedup over RECOMP at much higher accuracy.

Table 6: Walltime (s) across context lengths. Methods are grouped into preprocessing (top) and KV-cache (bottom).

Method	16k	32k	64k	128k	256k
SALT	**1.92**	**2.61**	4.01	**5.99**	**11.51**
EXIT	5.83	16.30	60.60	OOM	OOM
CPC	3.24	5.44	10.13	21.67	42.25
RECOMP	2.00	2.29	3.34	6.27	11.95
Sentinel	2.32	2.81	4.02	6.49	12.03
FastKV	1.73	2.35	4.22	10.77	34.29
SnapKV	2.00	3.08	6.43	18.13	61.75
DuoAttention	2.50	3.24	5.32	11.72	33.40
SentenceKV	10.43	17.06	31.20	63.80	64.40

D Efficiency

Table 7: Peak GPU memory (GB) across context lengths. Methods are grouped into preprocessing (top) and KV-cache (bottom).

Method	16k	32k	64k	128k	256k
SALT	**16.70**	**17.90**	**20.20**	**23.30**	**32.80**
EXIT	23.70	45.40	73.20	OOM	OOM
CPC	20.49	23.13	28.46	38.00	58.51
RECOMP	18.50	21.30	27.20	37.70	60.40
Sentinel	19.10	21.40	26.30	33.90	54.10
FastKV	17.37	19.39	23.43	31.52	47.67
SnapKV	17.57	19.79	24.20	33.12	50.88
DuoAttention	16.80	17.85	19.85	23.85	31.86
SentenceKV	22.98	28.50	39.80	60.73	60.89

E Hardware

Normalizing each run to 100% exposes a distinct shift in the component breakdown that is obscured by looking at absolute latency alone. For Ampere,

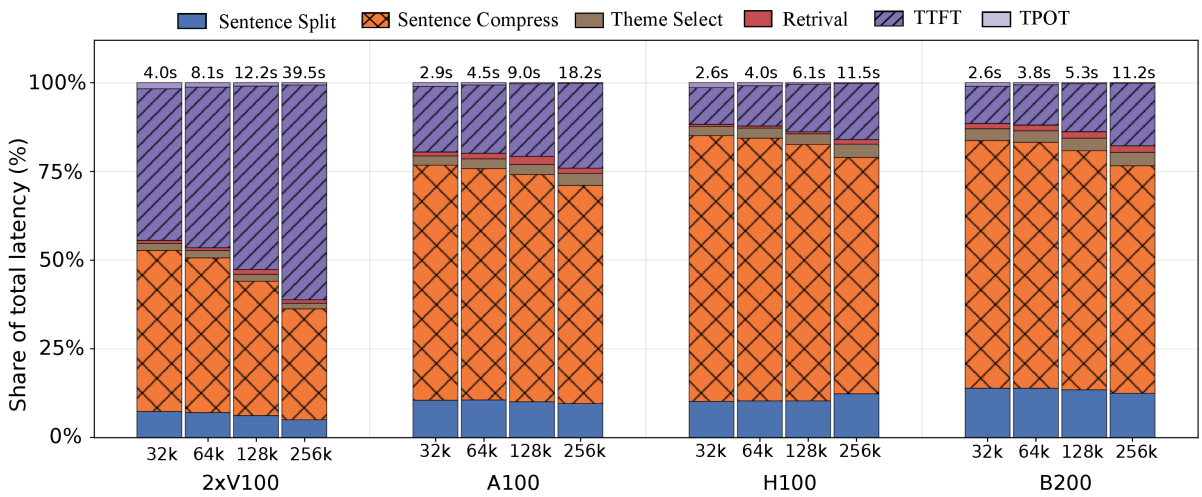


Figure 8: End-to-end latency and TPOT of SALT on Llama-3.1-8B-Instruct across different NVIDIA GPUs and raw-context lengths at a 20% retention budget, normalized so each bar sums to 100%.

Hopper, and Blackwell architectures, TTFT consistently accounts for roughly a third of the total time, while preprocessing forms the next largest segment and per-token decode fills the remainder.

The legacy V100 hardware breaks this pattern because TTFT becomes the dominant non-compression component, outlasting preprocessing at every context length. This divergence is likely architectural rather than algorithmic. The Volta generation lacks native bf16 support and cannot utilize the FlashAttention kernels designed for the tensor cores found in Ampere and subsequent architectures. As a result, the post-compression prefill attention must run in fp16 along an unoptimized execution path. Conversely, the compression phase is heavily dominated by lighter scoring and selection operations, making it relatively unaffected by these hardware limitations. This explains why the compression phase occupies a larger percentage share on newer hardware even as its absolute processing time decreases.

F Use of AI Assistants

Following the ACL policy on AI writing assistance, we used AI assistants based on large language models in a limited capacity during the preparation of this work. We restricted their role to editorial help such as grammar correction, rephrasing for clarity, and LaTeX formatting, along with auxiliary coding support like scripts for evaluation pipelines, plotting routines, and debugging. The research contributions in this paper, including the method design, experimental protocol, analysis, and interpretation of results, are entirely our own work. We did not use AI assistants to generate scientific claims, develop the core method, or produce experimental results. We reviewed and edited any AI-assisted text, and we take full responsibility for the final content.