SCOPE: STORED CHUNK-LEVEL OFFLINE PRIORS FOR EFFICIENT LONG-CONTEXT INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-context inference for large language models is limited by the repeated online scoring used to select important key-value (KV) blocks. We introduce SCOPE (Stored Chunk-Level Offline Priors), an efficient chunk-level method that reuses offline content priors across queries within a deployment domain. A store of offline chunk scores is built in the scheme of exponential moving average (EMA). Then unseen chunks are predicted with a lightweight ridge regressor, and finally intra- and inter-chunk refinement is applied at inference time with constant cost. This design replaces the dense scoring path of Flash Sparse Attention (FSA) with an expected-$\mathcal{O}(1)$ hash-table lookup per chunk followed by lightweight refinement. Although SCOPE uses static routing instead of the original query-dependent selector, it is still compatible with FSA's sparse-kernel interface. SCOPE adds no trainable parameters and requires only a small amount of auxiliary storage. Its store construction and lookup path can also run on CPUs alongside GPU computation, reducing inference latency without extending GPU pretraining. On Qwen-3.5-2B and Llama-3.2-1B, SCOPE improves median latency in 31 of 32 released-backbone cells, with average median and P99 reductions of 11.5% and 12.8%, respectively. On the larger Qwen2.5-7B model with a 32K context, SCOPE achieves a 15.3% average median-latency reduction. These results show that offline chunk priors provide an efficient alternative to repeated online scoring for long-context inference.

1 INTRODUCTION

The quadratic cost of self-attention is the main bottleneck of long-context LLM inference. Block-sparse attention is now the standard remedy. Each query allocates attention to only a top-n subset of key-value blocks, picked by a learned selector. Native Sparse Attention (NSA, Yuan et al., 2025) and its hardware-aligned version, Flash Sparse Attention (FSA, Yan et al., 2026), are representative examples of this design.

The scoring step for selecting top-n blocks used to be treated as a minor part of the kernel, but it has become the dominant computational cost. FSA's profile shows that token selection still takes up to 79% (mean 65%) of attention time after kernel-level tuning. Prefill speedup over NSA is only about $1.11\times$, and decoding latency is not improved. The real bottleneck is therefore no longer how attention is computed, but how important tokens or blocks can be identified quickly.

Existing methods try to make this scoring step cheaper. Examples include learned Hamming signatures (Desai et al., 2025), distilled gating networks (Gao et al., 2025b), per-token learned predictors (Akhauri et al., 2025), and per-head k-means clustering (Liu et al., 2025). These methods share the use of model-derived scores, which are computed online for the current query on every forward pass. A natural question is: can we change both the score source and timing with an offline content prior? We study this question for domain-specific inputs, rather than assuming query-invariant chunk rankings in advance.

To address this question, we propose SCOPE (Stored Chunk-Level Offline Priors). Its core idea is simple: **write during training, read during inference**. During training, SCOPE computes heuristic chunk scores from a domain corpus and stores their EMA values into an offline hash store. The chunk content is used as the key. For chunks absent from the store, a lightweight ridge regressor predicts the score. At inference time, SCOPE replaces FSA's online dense scoring kernel with an $\mathcal{O}(1)$ hash

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.

lookup, followed by lightweight intra-chunk and inter-chunk refinement. FSA's sliding-window and selected-attention branches remain unchanged.

SCOPE differs from prior work in three ways. (i) When a chunk is present in the hash store, the scoring computation is removed entirely, not just made faster. (ii) The lookup is keyed by chunk content, not by the query, so one score is shared across all heads and layers. (iii) The score replacement and FSA's kernel-level optimizations are independent of each other and can be combined.

Our contributions are summarized as follows.

1. **Algorithm.** We introduce SCOPE, a chunk-level offline architecture that replaces NSA/FSA's online dense scoring with an $\mathcal{O}(1)$ parameter-free hash lookup. SCOPE adds no trainable parameters to the host model, and its store can be built on the CPU alongside GPU pretraining.
2. **Experiments.** We deploy SCOPE on three publicly released backbones: Qwen-3.5-2B (Qwen Team, 2026) and Llama-3.2-1B (Grattafiori et al., 2024) at 8K-16K tokens context, and Qwen2.5-7B (Yang et al., 2024) at 32K. Across the 32 released-backbone cells, SCOPE improves median latency in 31 cells, with average median and P99 reductions of 11.5% and 12.8%, respectively. The separate Qwen2.5-7B experiment at 32K reports latency only and shows a 15.3%average median-latency reduction. The advantage is small at 8K tokens context and clearly larger at longer contexts.
3. **Analysis.** We report an operator-level latency breakdown and ablate the ChunkStore, ridge regressor, and two-level refinement to examine their effects on average chunk scores and measured latency.

2 Related Work

NSA (Yuan et al., 2025) is the algorithmic basis of SCOPE. It uses three branches: compressed, selected, and sliding-window. The compressed branch produces the importance signal for top-n selection, and this signal is recomputed online again for every query. FSA (Yan et al., 2026) is the hardware-aligned version of NSA. It adds online-softmax and reduction kernels backed by HBM buffers. Even with these kernels, token selection still takes up to 79% of attention time, with a mean of 65%. The dense scoring path may run the risk of spilling over global memory on long sequences. In contrast, SCOPE replaces this scoring path and reuses the other FSA kernels as is.

HashAttention (Desai et al., 2025) is the closest precedent. It replaces online dense scoring with learned 32-bit Hamming signatures. The signatures must be maintained per token, per head, and per layer. The authors also report that per-model training does not transfer from 64K to 128K. SCOPE instead precomputes the score at chunk granularity. It costs about $1/16$ byte per token, runs on the CPU, and adds no trainable parameters.

Several methods replace NSA-style scoring with cheaper online operators. SeerAttention (Gao et al., 2025b) distills a 101M-parameter AttnGate. However, the pooled-Q×pooled-K matrix is still computed online, and the gate is retrained per host model. TokenButler (Akhauri et al., 2025) distills per-token, per-head, per-layer importance. Scoring at every decoding step costs about as much as dense attention. ClusterKV (Liu et al., 2025) runs per-head k-means with a per-step query-centroid inner product, which costs $\mathcal{O}(n_i CLd)$ per query. InfLLM-V2 (Zhao et al., 2025) cuts parameter cost by sharing KV projections, but it still needs training and still selects online. All four methods change *how* the score is computed. For comparison, SCOPE changes both the score source and when it is computed: an offline IDF-and-position heuristic replaces online model-derived scoring.

Other approaches are orthogonal to SCOPE and compatible with it. They cover alternative sparse-attention scoring (Yang et al., 2025c; Zhu et al., 2025), KV-cache compression (Li et al., 2024a; Cai et al., 2025), RAG front-ends (Lim et al., 2025; Asai et al., 2024), and speculative decoding (Huang et al., 2025; Yang et al., 2025b). SCOPE does not interact with any of them. We give the full positioning in Appendix D.4.

3 SCOPE

3.1 Fundamental Architecture

FSA (Yan et al., 2026) serves as our baseline. Its inference path has three stages: compressed-attention scoring, top-K block selection, and sparse kernel computation. SCOPE targets the first

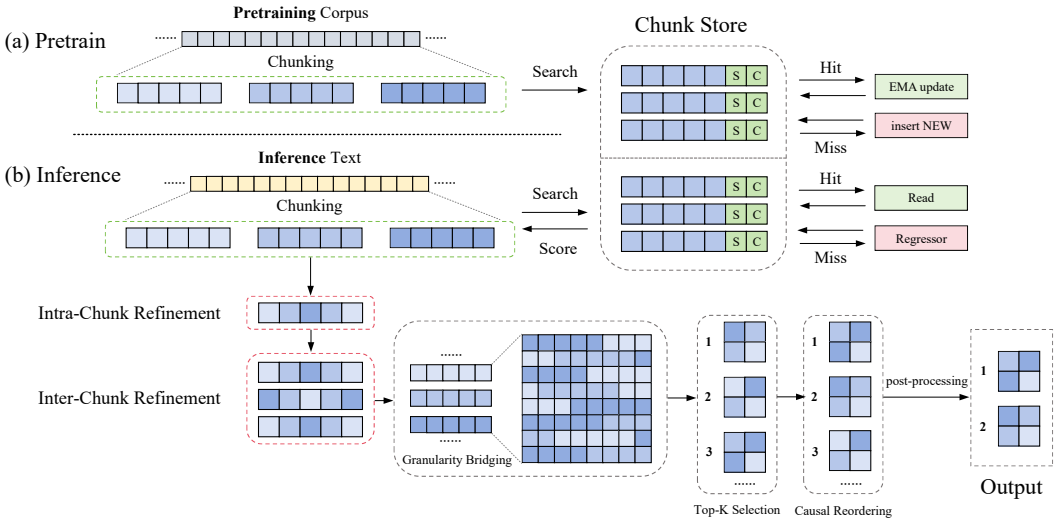


Figure 1: Two-stage SCOPE pipeline. (a) **Pretrain** (top): each chunk receives a heuristic score and is written to the store via EMA on hit or insert on miss. (b) **Inference** (bottom): each query chunk is looked up while misses fall back to the ridge regressor. The refined scores feed block aggregation and Top-K selection.

stage and leaves the other two unchanged. Therefore, SCOPE shares the same block size B, top-K, and sliding-window branch as FSA. The two methods use the same sparse-kernel interface, allowing us to compare their measured latency and quality under different routing mechanisms. The full positioning of FSA within the NSA family is given in Appendix D.4. SCOPE replaces FSA's dense compressed-attention scoring with an $\mathcal{O}(1)$ hash lookup. It targets domain-specific inference, where the chunk-content score is stable across queries.

There are four components in our pipeline. The semantic chunker $\mathcal{C}$ splits text into chunks. The hash store $\mathcal{H} : c \mapsto (s, \kappa)$ holds an EMA (Exponential Moving Average) score and a hit count, where c is the chunk content, s is its stored importance score, and κ is its hit count. The regression predictor R handles cold misses. A two-level refinement module then adjusts the prior scores using local context. All four components are assembled by the SCOPE routing wrapper, which plugs into the FSA sparse and sliding-window branches without touching the sparse kernel.

Figure 1 shows the data flow. During pretraining, each chunk receives a heuristic score and is written to the store. At inference time, the system looks up each chunk by content. If the chunk has already existed in the store, the lookup succeeds and the prestored score is used directly. If the chunk is new, the regression predictor R produces a score from a small set of hand-crafted features. The refined scores then drive max-pooled block aggregation and top-K selection.

3.2 Store Construction

This subsection describes how SCOPE builds the chunk-level hash store $\mathcal{H} : c \mapsto s$. The full pipeline runs on the CPU only, so it can run in parallel with GPU pretraining. The pipeline has four stages: corpus extraction, semantic chunking, two-pass scanning, and regression predictor training. We summarize each stage below and defer engineering details to Appendix D.

Formally, given a domain corpus $\mathcal{D} = \{T_1, \ldots, T_N\}$, each document T is segmented into chunks $\{C_1, \ldots, C_n\}$. The store learns the mapping $\mathcal{H}(C_i) \in [0, 1]$.

Corpus extraction and semantic chunking. The builder reads the domain corpus, applies a fixed field priority, and produces a unified chunk-level prior space across wiki, QA, and dialogue formats. The chunker uses word boundaries recovered by a subword tokenizer and targets $\ell \in [4, 7]$ words per chunk, with a preferred length $\ell^\star = 5$. Symbols are collapsed into seven buckets $\mathcal{B}$, and an idempotent normalizer $\phi(\cdot)$ aligns textual variants. The full priority schedule, normalization steps, and chunker pseudo-code are listed in Appendix D. The $(4, 5, 7)$ values are fixed heuristic hyperparameters

for minimum, preferred, and maximum chunk length. The seven symbol buckets are predefined categories.

Two-pass scanning. Pass 1 collects chunk-level inverse document frequency (IDF):

$$\mathrm{IDF}(c) = \log\!\big(\tfrac{N+1}{\mathrm{df}(c)+1}\big) + 1. \tag{1}$$

Pass 2 computes a continuous score in $[0, 1]$ for each chunk. Let p_i be the position of C_i in its document and n the number of chunks in that document. The score reuses the IDF term of Eq. 1 and is computed as:

$$s(C_i, p_i, n) = 0.6 \cdot \min\!\big(\tfrac{\mathrm{IDF}(C_i)}{6},\, 1\big) + 0.4 \cdot \max\!\big(0.5,\, 1 - 0.5\,\tfrac{p_i}{n}\big). \tag{2}$$

For symbol buckets, the score is fixed at $s(C_i) = 0.1$. The IDF term saturates at $\mathrm{IDF} \geq 6$. The position term decays to 0.5 at the document end. This matches the empirical prior that informative content tends to appear early. For each occurrence of C_i, we denote the resulting current score before EMA aggregation by $s^{(i)}_{\text{cur}}$. It equals $s(C_i, p_i, n)$ for regular chunks and 0.1 for symbol buckets.

For each pair $(C_i,\, s^{(i)}_{\text{cur}})$ produced by this scoring rule, the store performs a hash lookup with C_i as the key. If $C_i \notin \mathrm{dom}(\mathcal{H})$, we insert $\big(\mathcal{H}(C_i),\, \kappa(C_i)\big) = (s^{(i)}_{\text{cur}},\, 1)$. Otherwise, we update with EMA:

$$\mathcal{H}(C_i) \leftarrow \alpha\, s^{(i)}_{\text{cur}} + (1 - \alpha)\, \mathcal{H}(C_i), \quad \kappa(C_i) \mathrel{+}= 1,$$

with smoothing factor $\alpha = 0.2$. Once the entry limit M is reached, the store stops accepting new chunks. Existing keys keep getting EMA updates. In the end, the store has converged to the high-frequency core of the domain, and the final size is bounded by $\mathcal{O}(M)$.

Regression predictor. To handle cold misses at inference time, we train an offline predictor $R : C_i \mapsto \hat{s}^{(i)}$. Its feature vector $\mathbf{f}(C_i) \in \mathbb{R}^7$ has seven dimensions:

$$\mathbf{f}(C_i) = \big(\log(|C_i| + 1),\ \overline{\mathrm{IDF}}(C_i),\ \rho^{(i)}_{\text{rare}},\ \mathbb{I}^{(i)}_{\text{num}}, \mathbb{I}^{(i)}_{\text{cap}},\ \rho^{(i)}_{\text{stop}},\ n^{(i)}_{\text{punct}}\big)^{\top}. \tag{3}$$

The seven features are described as follows: (1) $\log(|C_i| + 1)$, the log chunk length in tokens, used as a length prior; (2) $\overline{\mathrm{IDF}}(C_i)$, the mean token-level IDF, which captures lexical specificity; (3) $\rho^{(i)}_{\text{rare}}$, the fraction of rare tokens with $\mathrm{IDF} > \tau_{\text{idf}}$, which highlights chunks dominated by uncommon terms; (4) $\mathbb{I}^{(i)}_{\text{num}}$, an indicator that the chunk contains a number; (5) $\mathbb{I}^{(i)}_{\text{cap}}$, an indicator that the chunk contains a capitalized token, a proxy for named entities; (6) $\rho^{(i)}_{\text{stop}}$, the stopword ratio, which downweights filler-heavy chunks; and (7) $n^{(i)}_{\text{punct}}$, the punctuation count, which separates clausal chunks from noun-phrase chunks.

Supervision comes from the EMA scores of the most-hit chunks. The default model is a closed-form ridge regression with $\lambda = 10^{-3}$:

$$\mathbf{w}^{\star} = \big(\mathbf{X}^{\top}\mathbf{X} + \lambda \mathbf{I}\big)^{-1}\mathbf{X}^{\top}\mathbf{y},$$

solved via the Moore–Penrose pseudoinverse. The final prediction is clipped to $[0, 1]$. The predictor is saved together with the chunk store as a single file. The full pseudocode of the build pipeline is given in Algorithm 3 (Appendix D).

3.3 Inference Alternative

This subsection describes how the chunk-level hash store, built during SCOPE pretraining, replaces FSA's dense compressed-attention branch at inference time. The sparse branch and the sliding-window branch are still preserved by the FSA kernel on the GPU. The framework is not tied to a specific backbone.

The goal is to produce a top-K index tensor $\mathrm{topk_idx} \in \mathbb{Z}^{H_{kv} \times L \times K}$ that matches the FSA kernel interface, without any dense forward pass. The pipeline has four stages: prior collection, two-level refinement, granularity bridging with top-K selection, and padding with gate zeroing. The corresponding pseudocode is given in Algorithm 4 (Appendix D).

Prior collection. For each chunk $\{C_i\}_{i=1}^{n}$, we adopt a read-only lookup. Hit and miss outcomes are written to the same vector slot, so downstream code needs no branching. We call this unified

per-chunk value the prior score $s^{(i)}_{\text{prior}}$. It uses the stored offline score for a hit and a clipped regression prediction for a miss:

$$s^{(i)}_{\text{prior}} = \begin{cases} \mathcal{H}(C_i), & C_i \in \text{dom}(\mathcal{H}), \\ \text{Clip}\big(R(\mathbf{f}(C_i)),\, 0,\, 1\big), & C_i \notin \text{dom}(\mathcal{H}). \end{cases} \tag{4}$$

Both branches are already calibrated to $[0, 1]$ during pretraining.

Two-level refinement. Based on the prior vector in Eq. 4, we apply an intra-chunk correction and an inter-chunk correction. The intra-chunk step adds three CPU-computable features:

$$s^{(i)}_{\text{adj}} = \text{Clip}\Big(s^{(i)}_{\text{prior}} + \beta_1 f^{(i)}_{\text{rare}} + \beta_2 f^{(i)}_{\text{dom}} - \beta_3 \cdot \sigma\big(|C_i| - \ell^\star\big),\, 0,\, 1\Big), \tag{5}$$

with $\beta_1 = 0.10$, $\beta_2 = 0.15$, $\beta_3 = 0.05$ in our implementation. The correction is additive and small, so the global statistics in the prior are not changed much. The inter-chunk step then adds a neighbor reward and an isolation penalty:

$$s^{(i)}_{\text{ref}} = \text{Clip}\Big(s^{(i)}_{\text{adj}} + \gamma\, I \cdot \big[s^{(i-1)}_{\text{adj}} \geq \tau_h\big] + \gamma\, I \cdot \big[s^{(i+1)}_{\text{adj}} \geq \tau_h\big] - 0.1\, \mathbb{I}^{(i)}_{\text{iso}},\, 0,\, 1\Big), \tag{6}$$

with $\gamma = 0.08$ and $\tau_h = 0.6$. This step pushes top-K selection toward contiguous high-score intervals, since point-wise estimates without neighborhood modeling cause semantic fragmentation.

Granularity bridging and top-K selection. The chunker returns chunks of $\ell \in [4, 7]$ words, but the FSA kernel operates on KV blocks of $B = 64$ tokens. We use the token index set $\mathcal{I}_i$ to map chunk scores to block scores. Let $\mathcal{R}_b = [bB, (b{+}1)B)$ be the token span of block b. The mapping is max-pooling over the refined scores in Eq. 6:

$$\text{block_score}[b] = \max_{i:\, \mathcal{I}_i \cap \mathcal{R}_b \neq \varnothing} s^{(i)}_{\text{ref}}, \tag{7}$$

with 0 for uncovered blocks. We use max-pooling because a block deserves selection if any high-importance chunk falls inside it, whereas average pooling might dilute the signal. After max-pooling, top-K selection takes the $n_{\text{avail}} = \min(K, \lceil L/B \rceil)$ highest-scoring entries of block_score in Eq. 7. We then sort these indices in ascending order to preserve the causal mask:

$$\mathcal{T}_{\text{ord}} = \text{sort}_{\uparrow}\Big(\arg\text{top-}n_{\text{avail}}\, \mathbf{block_score}\Big). \tag{8}$$

Padding, broadcast, and gate zeroing. The FSA kernel expects a fixed length K, so we pad missing slots with -1. The padded vector is then broadcast along the head dimension H_{kv} and the query position dimension L, so all KV heads share the same top-K set. Finally, SCOPE sets the compressed-branch gate to $g_{\text{cmp}} \leftarrow 0$. The final attention output returned to the host model is given as:

$$O_{\text{SCOPE}} = g_{\text{slc}}\, O_{\text{slc}} + g_{\text{win}}\, O_{\text{win}}. \tag{9}$$

Here O_{slc} is the output of the selected-attention branch over the top-K blocks, O_{win} is the output of the sliding-window branch. The scalars g_{slc}, g_{win} and g_{cmp} are FSA's per-branch gates. Setting $g_{\text{cmp}} = 0$ removes the dense compressed branch from the output. The block ranking produced by SCOPE is now supplied by topk_idx, which feeds the selected-attention branch O_{slc}. The full padding and broadcast formula is given in Appendix D.

When SCOPE runs at inference time, the whole scoring path finishes on the CPU. SCOPE therefore replaces an expensive GPU kernel with a much cheaper CPU lookup.

Algorithm 1 gives a whole picture of SCOPE. The build stage and the inference stage are shown side by side, with low-level details omitted. The full pseudocode is in Appendix D.

4 Experimental Setup

We evaluate SCOPE against FSA on attention latency and language-modeling quality. The setup includes one self-built Transformer, two released LLM backbones, and a larger 7B model under longer context. We report both efficiency and quality metrics, then defer low-level implementation details and dataset statistics to the Appendix.

Algorithm 1 SCOPE: end-to-end overview.

// Stage A: offline build (CPU, runs in parallel with GPU pretraining)
IDF $\leftarrow$ CHUNKDF$(\mathcal{F}, \mathcal{C})$ ▷ Pass 1: chunk-level DF
for $T \in \mathcal{F}$ **do** ▷ Pass 2: heuristic score + EMA write
  $\mathbf{C} \leftarrow \mathcal{C}(T)$
  for $C_i \in \mathbf{C}$ **do**
    $s \leftarrow$ HEURISTIC$(C_i, \text{IDF}, p_i, n)$ ▷ $0.6\,\iota + 0.4\,\pi$ in Alg. 3
    $\mathcal{H}$.EMAINSERT(C_i, s, α)
  end for
end for
$R \leftarrow$ RIDGEFIT$(\mathcal{H}, \lambda)$ ▷ cold-miss predictor

// Stage B: inference (per forward pass)
$\mathbf{C} \leftarrow \mathcal{C}(\text{input})$
for $C_i \in \mathbf{C}$ **do**
  $s^{(i)}_{\text{prior}} \leftarrow \mathcal{H}$.LOOKUP$(C_i)$ **or** $R(C_i)$ on miss
end for
$\mathbf{s}_{\text{ref}} \leftarrow$ REFINE$(\mathbf{s}_{\text{prior}})$ ▷ 2-level; see Alg. 4
$\mathbf{block_score} \leftarrow$ MAXPOOL$(\mathbf{s}_{\text{ref}}, \{\mathcal{I}_i\}, B)$
topk_idx $\leftarrow$ PAD(SORT$_\uparrow$(TOPK$(\mathbf{block_score})), -1)$
$g_{\text{cmp}} \leftarrow 0$; feed topk_idx to FSA sparse kernel

Hardware. The self-built Transformer, Qwen-3.5-2B, and Llama-3.2-1B experiments run on a single NVIDIA RTX 4090 D (24.6 GB) paired with an Intel i9-14900K CPU. The Qwen2.5-7B 32K experiment runs on one NVIDIA A800 80GB GPU with 16 CPU cores, 80GB RAM, a 200GB system disk, and a 600GB data disk. No multi-GPU and no quantization are used. The full software stack is in Appendix B.4.

Corpora. For generic pretraining, we use SlimPajama (Soboleva et al., 2023) as the training set $\mathcal{M}$. For domain pretraining of SCOPE, we use four long-text medical corpora as $\mathcal{D}$: PubMed-Summarization, PubMed Abstracts, MedQuAD, and BI55. For evaluation, we use four held-out medical sources, chosen to cover four input formats: PubMed23n (long-text), Medical-Flashcards (short-text), MedDialog-EN (dialogue), and MedQA-USMLE (multiple-choice). Per-source record counts and tokenizer statistics are in Appendix B.3.

Metrics. MedLat is the median per-forward attention latency in milliseconds. P99 is the 99th-percentile per-forward attention latency in milliseconds. PPL is token-level perplexity. Loss is cross-entropy. Lower is better for all four metrics.

Experiments. We run five sets of experiments. (1) *Self-built Transformer (§5.1).* A 10-layer Transformer with hidden size 4096 is pretrained for $13{,}000$ steps under Full Attention, NSA, and FSA on $\mathcal{M}$. SCOPE is then built on $\mathcal{D}$. We report MedLat, P99, PPL, and loss on the four test sets. (2) *Released LLM transfer (§5.2).* We inject FSA / SCOPE into Qwen-3.5-2B and Llama-3.2-1B and compare their attention paths on the same test sets. (3) *Larger model and domain store (§5.3).* We evaluate Qwen2.5-7B at 32K and test domain-specific stores on held-out law and math data. (4) *Length sweep (§5.4).* We vary sequence length on the two smaller backbones to study how the latency gap scales with L. (5) *Ablation (§5.5).* We disable each of the three pluggable components in turn: chunk store, regression predictor, and two-level refinement. The full latency protocol is in Appendix B.5.

5 RESULTS

5.1 SELF-BUILT TRANSFORMER EVALUATION

The self-built 10-layer Transformer serves as an initial validation on a compact backbone. Table 1 summarizes the latency and quality results, with each number being averaged over three independent runs.

The MedLat reduction over FSA is 15.70% to 16.93%, with a mean of 16.07%. The P99 reduction is 14.75% to 15.51%. PPL is nearly unchanged across the four corpora ($|\Delta\text{PPL}| \leq 0.90$). Particularly, SCOPE is 0.53 PPL lower than FSA on MedDialog-EN. For each kernel call, the score+TopK stage

Table 1: Self-built Transformer results on four medical corpora. Gains are relative to FSA.

Dataset	MedLat↓					P99↓				
	Full Attn	NSA	FSA	SCOPE	Gain↑	Full Attn	NSA	FSA	SCOPE	Gain↑
PubMed23n	2432.37	2474.76	2203.82	**1852.43**	15.94%	2442.40	2483.19	2227.06	**1881.64**	15.51%
Medical-Flashcards	2433.28	2473.87	2200.88	**1855.47**	15.70%	2437.50	2490.31	2223.19	**1886.28**	15.15%
MedDialog-EN	2432.43	2474.00	2200.76	**1828.20**	16.93%	2436.28	2479.62	2220.87	**1893.39**	14.75%
MedQA-USMLE	2433.48	2471.77	2203.19	**1857.23**	15.70%	2437.29	2504.25	2223.39	**1882.76**	15.33%
Dataset	PPL↓					Loss↓				
	Full Attn	NSA	FSA	SCOPE	ΔPPL	Full Attn	NSA	FSA	SCOPE	ΔLoss
PubMed23n	678.37	579.15	**392.62**	392.78	+0.16	6.5197	6.3616	**5.9728**	5.9733	+0.0004
Medical-Flashcards	1084.49	960.87	**605.99**	606.89	+0.90	6.9889	6.8678	**6.4069**	6.4083	+0.0015
MedDialog-EN	1596.23	1273.02	952.61	**952.08**	−0.53	7.3754	7.1492	6.8592	**6.8587**	−0.0006
MedQA-USMLE	600.73	509.64	**374.06**	374.42	+0.36	6.3981	6.2337	**5.9244**	5.9254	+0.0010

Table 2: MedQA-USMLE accuracy change of SCOPE over FSA on the two transferred backbones (length-normalized LL, 500 questions).

Backbone	ΔAcc (pp)	ΔAcc/FSA
Qwen-3.5-2B	−0.20	−0.75%
Llama-3.2-1B	−1.00	−4.39%

of SCOPE runs at 0.19–0.20 ms, against 40.4 ms for FSA and 141.6 ms for NSA. The end-to-end ∼350 ms MedLat saving therefore comes from the scoring path, not from the dense attention kernel.

We give the per-operator breakdown in Appendix C.9, the hit and select ratios in Appendix C.8, and the extended analysis in Appendix D.5.

5.2 Released LLM Transfer Evaluation

We next test whether SCOPE transfers from the compact validation backbone to released LLM backbones without losing its speedup or its quality match against FSA, the hardware-aligned baseline. We run two backbones, Qwen-3.5-2B (Qwen Team, 2026) and Llama-3.2-1B (Grattafiori et al., 2024). Both use GQA 4:1 but differ in head dimension (256 vs. 64) and layer count (24 vs. 16). We sweep four medical corpora at four sequence lengths (8K, 12K, 14K, 16K), giving 16 cells per backbone and 32 cells overall.

Latency. SCOPE improves median latency in 31 of 32 cells; the exception is Qwen-3.5 at 12K on Med-Flashcards (−0.38% median, −0.87% P99). On Qwen-3.5 at 16K, the P99 latency drops by up to 34.54% (PubMed23n, 1153.17 → 754.85 ms), and the median by up to 19.44% (MedQA-USMLE). On Llama-3.2 at 16K, median reductions range from 19.56% to 20.51%, while P99 reductions range from 17.55% to 18.92%. Figure 2 shows the full latency grid.

Tail behavior. Across Llama-3.2 cells, the gap between MedLat and P99 reductions reaches 4.80 pp (MedQA-USMLE at 14K). On Qwen-3.5 at 16K, the P99 reduction is nearly twice the median. The larger KV head dimension of Qwen drives FSA's dense scoring kernel past the on-chip workspace, producing the tail spike. SCOPE uses a fixed-cost lookup path and avoids this scoring-kernel spill. We expand on this in §5.4.

Quality. On Qwen-3.5, it improves average PPL by 1.2524 at 8K and 0.8855 at 16K. By contrast, on Llama-3.2, average PPL increases by 0.8510 at 8K and 0.5689 at 16K; the largest increases occur on MedQA (2.90 → 5.00 at 8K; 2.92 → 4.87 at 16K; Table 20). Decision-level fidelity holds as well: on MedQA-USMLE, the four-way MCQ accuracy gap stays within 1 pp on both backbones (Table 2).

Overall, SCOPE substantially reduces attention latency across the tested backbones while keeping measured quality broadly comparable. Hit and select ratios, the full per-cell latency and PPL grid, and the MedQA-USMLE protocol are deferred to the Appendix.

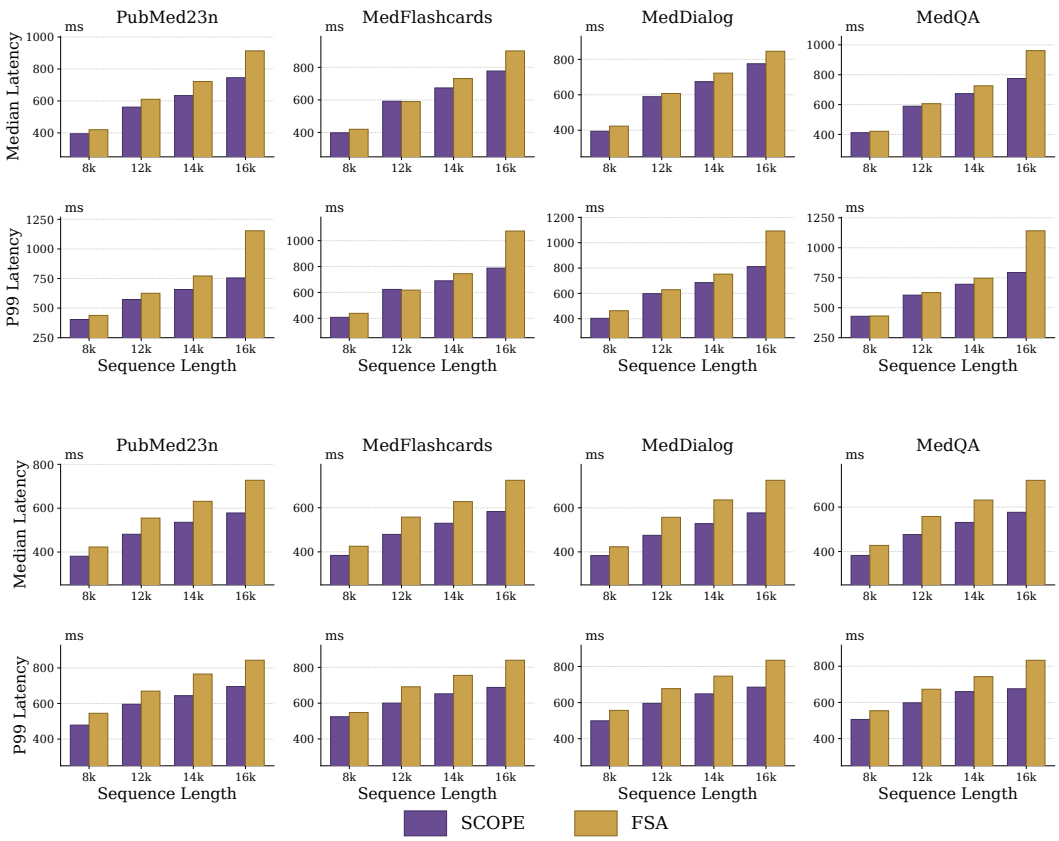


Figure 2: Median (top) and P99 (bottom) attention latency of FSA vs. SCOPE at 8K, 12K, 14K, and 16K on Qwen-3.5 (upper) and Llama-3.2 (lower).

Table 3: Qwen2.5-7B results at 32K context. Latency is reported in milliseconds. Lower is better.

Dataset	FSA MedLat	SCOPE MedLat	FSA P99	SCOPE P99
PubMed23n	3585.0	3025.9	3589.4	3033.3
Medical Flashcards	3584.3	3035.1	3586.0	3038.9
MedDialog-EN	3577.7	3041.7	3581.4	3046.8
MedQA-USMLE	3585.8	3033.0	3589.2	3034.7

5.3 Larger Model and Domain-Specific Store Evaluation

We next test whether the same offline-prior idea still helps under a larger model and a longer context. Table 3 reports Qwen2.5-7B (Yang et al., 2024) results at 32K on the four medical datasets. SCOPE reduces median latency by 15.0–15.6% and P99 latency by 14.9–15.5% relative to FSA. The average median-latency reduction is 15.3% and the average P99 reduction is also about 15.3%.

We test domain-specific stores on held-out math and law at 8K and a fixed medical store on finance, law, and math at 16K (Tables 4 and 5).

5.4 Latency Reduction Trend

Figure 3 summarizes our sequence-length sweep on Qwen-3.5 and Llama-3.2. Most points average four datasets; partial sweeps use the available datasets, with n reported in Appendix Table 21. The speedup grows with sequence length on both backbones. Llama rises at a steady slope and saturates near 17K. Qwen shows a sharp surge in $[15\text{K}, 16\text{K}]$ and an unexpected drop after 16K.

Table 4: Domain-specific chunk-store evaluation on held-out non-medical data. Positive gain means SCOPE is faster than FSA.

Store/Test	FSA MedLat	SCOPE MedLat	Gain	Hit Ratio
MATH → MATH	430.1	418.4	+2.7%	96.6%
LexGLUE → Law	425.5	456.1	−7.2%	80.3%

Table 5: Cross-domain evaluation of a fixed medical chunk store with Llama-3.2-1B at 16K. Positive gain means SCOPE is faster than FSA.

Domain	FSA MedLat	SCOPE MedLat	Med Gain	P99 Gain	Hit Ratio
Finance	687.4	566.4	17.6%	15.1%	73.9%
Law	695.1	562.1	19.1%	14.7%	74.5%
Math	692.0	564.1	18.5%	17.5%	94.0%

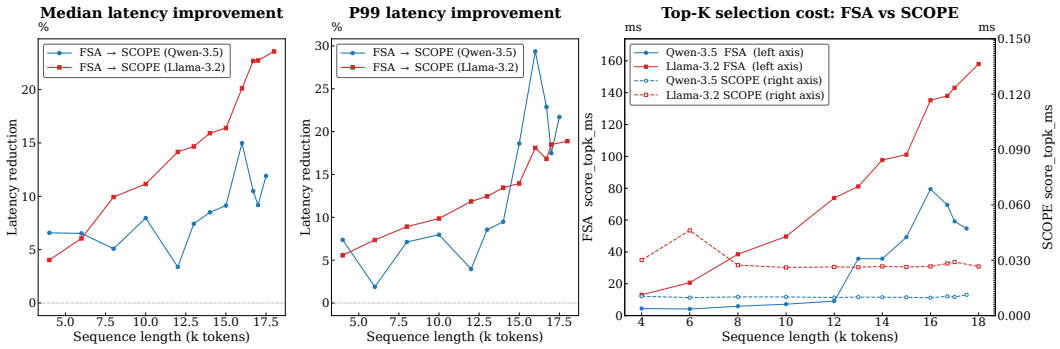


Figure 3: Latency reduction of SCOPE over FSA across 4–18K tokens, averaged over the four medical corpora. Left and middle: median and P99 reduction. Right: per-forward Top-K cost (FSA on the left axis, SCOPE on the right axis).

On Qwen-3.5 with our RTX 4090D setup, FSA's score+Top-K time rises from 49.26 ms at 15K to 79.44 ms at 16K, consistent with a configuration-specific kernel-resource threshold. The 18K point is a single PubMed run with FSA's init+local fallback, so it follows a different execution path (Appendix Table 21). By contrast, SCOPE's score+Top-K time remains nearly constant as context length grows, giving more predictable scaling.

5.5 Ablation Study

The Qwen-3.5 16K ablation covers ChunkStore lookup, regression prediction, and two-level refinement; AvgScr, P99, and MedLat are primary metrics, with PPL/Loss as guardrails. Appendix Table 15 reports full results. Removing ChunkStore raises AvgScr (0.615 → 0.989) but lowers mean MedLat/P99 (786.6/851.4 → 755.8/783.1 ms). Random misses raise MedQA P99 to 1079.0 ms; constant misses lower mean P99 to 812.2 ms. Disabling refinement leaves PPL nearly unchanged ($\Delta = -0.001$) and raises mean MedLat to 813.6 ms.

6 Conclusion

FSA's dense online scoring limits sparse attention. SCOPE replaces online chunk scores with offline priors while retaining FSA's attention kernels. Across tested backbones and medical lengths, latency falls in 31/32 cells by about 11.5% on average. Qwen2.5-7B reaches a 15.3% reduction at 32K. Our future work is to extend SCOPE to longer contexts and broader deployment domains while co-designing the remaining sparse-attention branches with the offline path.

AI USE STATEMENT

Generative AI tools were used to aid and polish the writing and to draft sections of this paper. The authors reviewed the AI-assisted text and take responsibility for the final content of this manuscript.

REFERENCES

Shantanu Acharya, Fei Jia, and Boris Ginsburg. Star attention: Efficient LLM inference over long sequences. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 356–371, 2025.

Yash Akhauri, Ahmed F. AbouElhamayed, Yifei Gao, Chi-Chih Chang, Sameh Gobriel, et al. Tokenbutler: Token importance is predictable, 2025. arXiv:2503.07518. Version 2, May 2026.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avi Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations (ICLR)*, pp. 9112–9141, 2024.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, et al. LongBench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3639–3664, 2025.

Payman Behnam, Yaosheng Fu, Ritchie Zhao, Po-An Tsai, Zhiding Yu, et al. Rocketkv: Accelerating long-context LLM inference via two-stage KV cache compression. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 3358–3392, 2025.

Asma Ben Abacha and Dina Demner-Fushman. A question-entailment approach to question answering. *BMC Bioinformatics*, 20(1):511, 2019.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. In *Proceedings of the 2025 Conference on Language Modeling (COLM 2025)*, 2025.

Ziyi Cao, Qingyi Si, Jingbin Zhang, and Bingquan Liu. Sparse attention across multiple-context kv cache. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(36):30165–30173, 2026.

Aditya Desai, Shuo Yang, Alejandro Cuadron, Matei Zaharia, Joseph E. Gonzalez, et al. Hashattention: Semantic sparsity for faster inference. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 13402–13418, 2025.

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 38 (NeurIPS 2025)*, volume 38, pp. 113152–113188, 2025.

Yizhao Gao, Shuming Guo, Shijie Cao, Yuqing Xia, Yu Cheng, et al. Seerattention-r: Sparse attention adaptation for long reasoning, 2025a. arXiv:2506.08889.

Yizhao Gao, Zhichen Zeng, DaYou Du, Shijie Cao, Peiyuan Zhou, et al. Seerattention: Self-distilled attention gating for efficient long-context prefilling. In *Advances in Neural Information Processing Systems*, volume 38, pp. 55846–55869, 2025b.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. The llama 3 herd of models, 2024. arXiv:2407.21783.

Vivek Gupta, Prerna Bharti, Pegah Nokhiz, and Harish Karnick. Sumpubmed: Summarization dataset of pubmed scientific articles. In *Proceedings of the 59th annual meeting of the Association for Computational Linguistics and the 11th international joint conference on natural language processing: student research workshop*, pp. 292–303, 2021.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, et al. Ruler: What's the real context size of your long-context language models? In *Proceedings of the First Conference on Language Modeling (COLM 2024)*, 2024.

Kaiyu Huang, Hao Wu, Zhubo Shi, Han Zou, Minchen Yu, et al. Adaspec: Adaptive speculative decoding for fast, slo-aware large language model serving. In *Proceedings of the 2025 ACM Symposium on Cloud Computing (SoCC '25)*, pp. 361–374, 2025.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, volume 37, pp. 52481–52515, 2024.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, et al. What disease does this patient have? a large-scale open domain question answering dataset from medical exams, 2020. arXiv:2009.13081.

Jiajie Jin, Xiaoxi Li, Guanting Dong, Yuyao Zhang, Yutao Zhu, et al. Hierarchical document refinement for long-context retrieval-augmented generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3502–3520, 2025.

Jordan Juravsky, Bradley Brown, Ryan Ehrlich, Daniel Y. Fu, Christopher Ré, et al. Hydragen: High-throughput llm inference with shared prefixes, 2024. arXiv:2402.05099.

Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, et al. Babilong: Testing the limits of llms with long context reasoning-in-a-haystack. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024) – Datasets and Benchmarks Track*, volume 37, pp. 106519–106554, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, et al. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP 2023)*, pp. 611–626, 2023.

Xunhao Lai, Jianqiao Lu, Yao Luo, Yiyuan Ma, and Xun Zhou. Flexprefill: A context-aware sparse attention mechanism for efficient long-sequence inference, 2025. arXiv:2502.20766.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, et al. Snapkv: Llm knows what you are looking for before generation. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, volume 37, pp. 22947–22970, 2024a.

Zixuan Li, Jing Xiong, Fanghua Ye, Chuanyang Zheng, Xun Wu, et al. Uncertaintyrag: Span-level uncertainty enhanced long-context modeling for retrieval-augmented generation, 2024b. arXiv:2410.02719.

Woosang Lim, Zekun Li, Gyuwan Kim, Sungyoung Ji, HyeonJung Kim, et al. Macrag: Compress, slice, and scale-up for multi-scale adaptive context rag, 2025. arXiv:2505.06569.

Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. Clusterkv: Manipulating llm kv cache in semantic space for recallable compression. In *Proceedings of the 62nd ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2025.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, et al. Moba: Mixture of block attention for long-context llms. In *Advances in Neural Information Processing Systems*, volume 38, pp. 17790–17815, 2025a.

Wensheng Lu, Keyu Chen, Ruizhi Qiao, and Xing Sun. Hichunk: Evaluating and enhancing retrieval-augmented generation with hierarchical chunking, 2025b. arXiv:2509.11552.

National Center for Biotechnology Information. Pubmed, 2023.

Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, et al. Mooncake: A kvcache-centric disaggregated architecture for llm serving, 2024. arXiv:2407.00079.

Qwen Team. Qwen3.5: Towards native multimodal agents, 2026.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, et al. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, volume 37, 2024.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, et al. Slimpajama: A 627b token cleaned and deduplicated version of redpajama, 2023.

Yutao Sun, Tianzhu Ye, Li Dong, Yuqing Xia, Jian Chen, et al. Rectified sparse attention, 2025. arXiv:2506.04108.

Chihiro Taguchi, Seiji Maekawa, and Nikita Bhutani. Efficient context selection for long-context QA: No tuning, no iteration, just adaptive-k. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20105–20130, 2025.

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

Zicong Tang, Shi Luohe, Zuchao Li, Baoyuan Qi, Liu Guoming, et al. SpindleKV: A novel KV cache reduction method balancing both shallow and deep layers. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28428–28442, 2025.

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL@PLDI 2019)*, pp. 10–19, 2019.

Haoxuan Wang and Chen Wang. Hierasparse: Hierarchical semi-structured sparse kv attention, 2026. arXiv:2604.16864.

Jeffrey Willette, Heejun Lee, and Sung Ju Hwang. Delta attention: Fast and accurate sparse attention inference by delta correction. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025)*, volume 38, pp. 12052–12080, 2025.

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 (ICML)*, 2025.

Minghao Yan, Saurabh Agarwal, and Shivaram Venkataraman. Decoding speculative decoding. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 6460–6473, 2025.

Ran Yan, Youhe Jiang, Zhuoming Chen, Haohui Mai, Beidi Chen, et al. Fsa: An alternative efficient implementation of native sparse attention kernel. In *The Fourteenth International Conference on Learning Representations*, 2026. Also available as arXiv:2508.18224.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, et al. Qwen2.5 technical report, 2024. arXiv:2412.15115.

Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. Kvlink: Accelerating large language models via efficient kv cache reuse. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025)*, volume 38, pp. 133797–133824, 2025a.

Mingyu Yang, Jae-Young Choi, Kihyo Moon, Minsung Jang, and Eunjoo Jeon. Dsde: Dynamic speculative decoding with kld stability for real-world serving, 2025b. arXiv:2509.01083. Accepted for presentation at the IEEE BigData 2025 Workshop (Special Session on Intelligent Data Mining).

Shang Yang, Junxian Guo, Haotian Tang, Qinghao Hu, Guangxuan Xiao, et al. Lserve: Efficient long-sequence llm serving with unified sparse attention. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 7, 2025c.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, et al. 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)*, volume 1, pp. 23078–23097, 2025.

Guangtao Zeng, Wenmian Yang, Zeqian Ju, Yue Yang, Sicheng Wang, et al. Meddialog: Large-scale medical dialogue datasets. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 9241–9250, 2020.

Qingfei Zhao, Ruobing Wang, Yukuo Cen, Daren Zha, Shicheng Tan, et al. LongRAG: A dual-perspective retrieval-augmented generation paradigm for long-context question answering. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 22600–22632, 2024.

Weilin Zhao, Zihan Zhou, Zhou Su, Chaojun Xiao, Yuxuan Li, et al. Infllm-v2: Dense-sparse switchable attention for seamless short-to-long adaptation, 2025. arXiv:2509.24663.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, et al. Sglang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, volume 37, pp. 62557–62583, 2024.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, et al. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210, 2024.

Qianchao Zhu, Jiangfei Duan, Chang Chen, Siran Liu, Xiuhong Li, et al. Sampleattention: Near-lossless acceleration of long context llm inference with adaptive structured sparse attention. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 7, 2025.

A Limitations

SCOPE rests on a few assumptions that we now state explicitly. First, the offline store assumes that chunk-content importance is roughly stable within a deployment domain. The law-store result shows that domain drift can reduce selection efficiency even when the hit ratio remains reasonably high. This narrows the latency margin and can push the operating point back toward online scoring. Second, building the store requires a representative corpus that is available offline at training time. Cold-start deployments, where no in-domain text and no calibration pass is available, are outside the scope of SCOPE and would need a bootstrap procedure that we do not study. Third, the 32K validation is limited to Qwen2.5-7B. We do not evaluate 64K, 128K, or million-token contexts, where memory pressure may require additional engineering. Finally, SCOPE leaves the sliding-window and selected-attention branches of NSA/FSA untouched, and it inherits whatever constraints those kernels carry on new hardware. Co-designing the surviving branches with the offline path is left to future work. Long-context-benchmark sweeps and the extended positioning against RAG front-ends and speculative decoding are discussed in Appendix D.4. The 8K-16K quality tests use cyclically tiled short documents; evaluation on non-repeated long inputs remains future work.

B Implementation Details

B.1 Hyperparameters and Configurations

This subsection collects every numerical knob that controls the SCOPE pipeline and the backbones used in §5. Table 6 lists the SCOPE-specific hyperparameters, grouped by stage of Algorithms 3 and 4. Table 7 reports backbone shape and sparse-attention configuration for the from-scratch 10-layer Transformer, Qwen-3.5-2B (Qwen Team, 2026), and Llama-3.2-1B (Grattafiori et al., 2024). Table 8 gives the separate Qwen2.5-7B (Yang et al., 2024) configuration used for the 32K experiment.

B.2 Field-Extraction Priority Schedule

The medical training shards expose textual content under inconsistent field names: PubMed uses `article/abstract`, MedDialog-EN uses `utterances`, and MedQuAD uses `Question/Answer`. To unify these into the chunk-level prior space of §3.2, the build pipeline applies a deterministic two-stage extraction schedule listed in Table 9.

Stage 1 walks the *concatenation groups* in order. For each group, if every member field is present (case-insensitive), the values are joined by a newline and returned. Stage 2 is reached only if no group fires. It walks the *singleton priority list* and returns the first non-empty value. As a final fallback, all

Table 6: SCOPE-specific hyperparameters used in all reported runs. The five blocks correspond to the offline store builder, the heuristic chunk scorer, the regression predictor, the two-level refinement, and the granularity bridge / Top-K post-processing.

Stage	Symbol / Flag	Value	Description and source
(i) Offline store builder (`scripts/build_dadcs_store_v3.py`)			
EMA write	α	0.2	EMA smoothing rate for repeated chunks
Store cap	M (`-max-store-entries`)	1,800,000	Hard upper bound; once reached, miss-path insertion is disabled and only EMA updates remain
Token cap	L_{tok} (`-seq-len`)	40,000	Per-document token truncation before chunking
Character cap	L_{char} (`-max-doc-chars`)	200,000	Pre-tokenizer character truncation
Per-file doc cap	M_{doc} (`-max-docs-per-file`)	20,000	Mitigates single-source domination
Min input tokens	–	8	Documents shorter than 8 tokens are skipped
(ii) Heuristic chunk scorer (`score_chunk_heuristic`)			
IDF weight	–	0.6	Coefficient on $\min(\text{IDF}/6, 1)$
Position weight	–	0.4	Coefficient on $\max(0.5,\ 1 - 0.5\,p/n)$
IDF saturation	–	6	Cap above which $\text{IDF}/6 = 1$
Min position factor	–	0.5	Floor of the document-position term
Symbol-bucket score	–	0.1	Constant score for any chunk in $\mathcal{B}$
(iii) Regression predictor (`dadcs/core/regression_predictor.py`)			
Ridge regulariser	λ	10^{-3}	L_2 strength in $(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}$
Feature dimension	$\lvert\mathbf{f}(c)\rvert$	7	$(\log\lvert c\rvert, \overline{\text{IDF}}, \rho_{\text{rare}}, \mathbb{I}_{\text{num}}, \mathbb{I}_{\text{cap}}, \rho_{\text{stop}}, n_{\text{punct}})$
Rare-word IDF cut	τ_{idf}	3.0	ρ_{rare} counts w with $\text{IDF}(w) > \tau_{\text{idf}}$
Training subset	top-50K	50,000	Most-hit chunks (by κ) used as supervision
Min training size	–	200	Below this, regression training is skipped
Symbol filter	–	active	Keys matching `"<...>"` are excluded
(iv) Two-level refinement (`dadcs/core/two_level_refinement.py`)			
Rare-word weight	β_1	0.10	Intra-chunk additive on f_{rare}
Domain weight	β_2	0.15	Intra-chunk additive on f_{dom}
Length-redundancy	β_3	0.05	Intra-chunk subtractive on $\sigma(\lvert C\rvert - \ell^\star)$
Neighbour reward	$\gamma_1 = \gamma_2$	0.08	Inter-chunk continuity bonus per side
High-score gate	τ_h	0.6	Threshold for both reward and isolation
Isolation penalty	–	0.1	Subtractive on isolated high-score chunks
Target chunk len.	$\ell^\star$	5 words	Sigmoid center in the length term
(v) Granularity bridge & Top-K post-processing (`dadcs/module/dadcs_attention.py`)			
Chunker bounds	$(\ell_{\min}, \ell^\star, \ell_{\max})$	$(4, 5, 7)$ words	Word-count window for the semantic chunker
Top-K ratio	`topk_ratio`	0.4	Diagnostic Select-Ratio target for Table 19
Aggregation	–	max-pool	block_score$[b] = \max_{i \in b} s_{\text{ref}}^{(i)}$
Sentinel value	–	-1	Padding for unfilled Top-K slots
Compressed gate	g_{cmp} at inference	0	Structurally bypasses the dense scoring branch

Table 7: Backbone shape and sparse-attention configuration for the three host models. The first block lists backbone-shape parameters fixed by the released config files; the second block lists the FSA / SCOPE sparse-branch parameters injected at inference.

Block	**Parameter**	**Self-built 10-layer**	**Qwen-3.5-2B**	**Llama-3.2-1B**
Backbone shape (released config / pretraining script)				
Backbone	`hidden_size`	4096	2048	2048
Backbone	`num_q_heads`	32	8	32
Backbone	`num_kv_heads`	8	2	8
Backbone	GQA ratio (Q:KV)	4:1	4:1	4:1
Backbone	`head_dim`	128	256	64
Backbone	`num_hidden_layers`	10	24	16
Backbone	FFN multiplier	1.5	3.0†	4.0†
Backbone	dtype	bf16	bf16	bf16
Backbone	evaluation `seq_len`	49,152	{8K, 12K, 14K, 16K}	{8K, 12K, 14K, 16K}
Sparse-attention configuration (FSA / SCOPE injection)				
Sparse	`kernel_size`	32	32	32
Sparse	`kernel_stride`	16	8	8
Sparse	`block_size` (B)	64	64	64
Sparse	`topk`	12	12	12
Sparse	`init_blocks`	1	1	1
Sparse	`local_blocks`	2	2	2
Sparse	`window_size`	512	512	512

† FFN multiplier reported as `intermediate_size/hidden_size`: Qwen-3.5-2B uses $6144/2048 = 3.0$, Llama-3.2-1B uses $8192/2048 = 4.0$. These two ratios are not stated in the main text and are derived directly from the released `config.json` of each backbone.

Table 8: Qwen2.5-7B configuration for the 32K experiment. The sparse branch keeps the same FSA/SCOPE semantics as the smaller transferred backbones.

Parameter	Value
Backbone	Qwen2.5-7B
dtype	bf16
evaluation `seq_len`	32K
GPU	1× NVIDIA A800 80GB
CPU / RAM	16 CPU cores / 80GB RAM
System / data disk	200GB / 600GB
`block_size` (B)	64
`topk`	12
`init_blocks`	1
`local_blocks`	2
`window_size`	512

`str`-typed fields are concatenated; this case is recorded as `<fallback-concat>` in Table 11. Two design choices matter for reproducibility. First, concatenation groups are tested before the singleton list, so QA-style records keep a (question, answer) pair as a single contiguous chunk. Second, the singleton list places `article`/`text`/`content`/`abstract` ahead of `question`/`answer`, so a PubMed record with both an `article` body and an `abstract` contributes the full body, not the abstract alone.

Table 9: Field-extraction schedule used by the build pipeline. Top: concatenation groups, tested first; all members must be present. Bottom: singleton priority list, case-insensitive; first non-empty match wins.

Concatenation groups (Stage 1, in order)
1. `Prompt` + `Completion`
2. `prompt` + `completion`
3. `instruction` + `input` + `output`
4. `Questions` + `Answers`
5. `question` + `answer`
6. `description` + `utterances`

Singleton priority list (Stage 2, in order)			
1.	`article`	14.	`body`
2.	`text`	15.	`Answers`
3.	`content`	16.	`Answer`
4.	`abstract`	17.	`Questions`
5.	`answer`	18.	`Question`
6.	`long_answer`	19.	`Text`
7.	`passage`	20.	`Content`
8.	`context`	21.	`Prompt`
9.	`question`	22.	`prompt`
10.	`dialogue`	23.	`Completion`
11.	`output`	24.	`completion`
12.	`input`	25.	`description`
13.	`instruction`	26.	`Description`

B.3 Training and Evaluation Corpora

This subsection lists every dataset used to populate the chunk store and to drive the long-context evaluation in §5. For *characters per record* the $L_{\text{char}} = 2\times10^5$ cap is binding only for a small tail of long PubMed articles ($\sim$50 docs per shard); all other counts are reported on the unmodified raw text.

Licenses. All backbones and corpora used in this work are publicly available for research purposes under the following licenses. **Backbones:** Qwen-3.5-2B is released under Apache 2.0; Llama-3.2-1B is released under the Llama 3.2 Community License. **Training corpora:** SlimPajama (Soboleva et al., 2023) is released under Apache 2.0; MedQuAD (Ben Abacha & Demner-Fushman, 2019) is released under CC BY 4.0; PubMed Abstracts and PubMed-Summarization (Gupta et al., 2021) are derived from the NCBI PubMed corpus, distributed under NCBI's terms of use (U.S. public-domain metadata, with copyright on individual abstracts retained by their original publishers); BI55 `medtext_2` is distributed under the terms of its Hugging Face dataset card. **Evaluation corpora:** PubMed23n (National Center for Biotechnology Information, 2023) is distributed by NCBI under the same PubMed terms; MedDialog-EN (Zeng et al., 2020) and Medical-Flashcards follow the terms of their respective dataset cards; MedQA-USMLE (Jin et al., 2020) is released under the MIT License. Our use of every artifact is consistent with its intended research purpose. The SCOPE code and trained checkpoints we release will be distributed under Apache 2.0; checkpoints derived from Qwen-3.5-2B and Llama-3.2-1B inherit any redistribution constraints from their respective base-model licenses.

Field extraction. Records are harvested by the two-stage schedule of Appendix B.2.

Training corpora. The training mixture is four medical sources plus a SlimPajama (Soboleva et al., 2023) subset (Table 10). The per-file cap 2×10^4 binds for MedQuAD (Ben Abacha & Demner-Fushman, 2019), every PubMed-Summarization shard (Gupta et al., 2021), and every SlimPajama shard. Token columns are reported separately for the two tokenizers used downstream.

Table 10: Training corpora used to populate the SCOPE chunk store. "Raw" is the number of records with non-empty text after field extraction; "After cap" is what remains after the per-file 2×10^4 cap. Char and token sums† are over the raw records.

Source	Field(s) used	Raw	After cap	Chars†	Tok (Qwen)†	Tok (Llama)†
PubMed Abstracts	`text`	100	100	1.47×10^5	3.17×10^4	3.09×10^4
MedQuAD	`Questions+Answers`	47 441	20 000	2.38×10^7	5.08×10^6	5.04×10^6
BI55 medtext_2	`Prompt+Completion`	1 412	1 412	9.34×10^5	1.89×10^5	1.87×10^5
PubMed-Summ. 0	`article` (97.6%) / `abstract`	23 985	20 000	4.19×10^8	9.54×10^7	9.25×10^7
PubMed-Summ. 1	`article` (97.6%) / `abstract`	23 985	20 000	4.17×10^8	9.47×10^7	9.18×10^7
PubMed-Summ. 2	`article` (97.6%) / `abstract`	23 985	20 000	4.12×10^8	9.38×10^7	9.09×10^7
PubMed-Summ. 3	`article` (97.8%) / `abstract`	23 985	20 000	4.23×10^8	9.60×10^7	9.31×10^7
PubMed-Summ. 4	`article` (97.6%) / `abstract`	23 984	20 000	4.19×10^8	9.52×10^7	9.24×10^7
SlimPajama‡	`text`	$14\times50\,000$	$14\times20\,000$	3.02×10^9	6.85×10^8	6.58×10^8
Medical total	—	**168 877**	**121 512**	$\mathbf{2.12\times10^9}$	$\mathbf{4.80\times10^8}$	$\mathbf{4.66\times10^8}$
SlimPajama total	—	**700 000**	**280 000**	$\mathbf{3.02\times10^9}$	$\mathbf{6.85\times10^8}$	$\mathbf{6.58\times10^8}$
Combined storage input	—	**868 877**	**401 512**	$\mathbf{5.14\times10^9}$	$\mathbf{1.16\times10^9}$	$\mathbf{1.12\times10^9}$

‡ The SlimPajama row aggregates 14 shards. Per-shard raw, kept, character-sum, and token-sum counts are within $\pm2\%$ of the shard mean, so only the 14-shard totals are reported. † Character and token sums are computed on the unfiltered text (no L_{tok} or L_{char} truncation, no per-file document cap), so they upper-bound the input the build pipeline sees. At build time the sequence-length limit is $L_{\text{tok}}=1024$ tokens and the document is clipped to $L_{\text{char}}=2\times10^5$ characters. The latter trims at most about 50 outlier docs per shard.

Evaluation corpora. Long-context benchmarking uses four held-out medical sources (Table 11): PubMed23n abstracts (National Center for Biotechnology Information, 2023), Medical-Flashcards, MedDialog-EN (Zeng et al., 2020), and MedQA-USMLE (Jin et al., 2020). Each source is capped at 500 documents (MedDialog-EN keeps its full 479-record set).

Table 11: Evaluation corpora. "Raw" is the number of records with non-empty text. "Eval" is the number of records actually fed to the benchmark, after the 500-record cap. Mean lengths are computed on the eval-capped subset.

Source	Field(s) used	Raw	Eval	$\overline{\lvert T\rvert_{\text{char}}}$	$\overline{\lvert T\rvert_{\text{tok}}^{\text{Qwen}}}$	$\overline{\lvert T\rvert_{\text{tok}}^{\text{Llama}}}$
PubMed23n (20 shards, 1–20)	`text`	261 124	20×500	$\sim$833−1159	$\sim$178−269	$\sim$176−266
e.g. `pubmed23n0001`	`text`	15 377	500	1158.79	268.82	265.47
e.g. `pubmed23n0008`	`text`	15 373	500	1048.50	234.43	231.10
e.g. `pubmed23n0019`	`text`	13 656	500	833.87	178.10	176.02
Medical-Flashcards	⟨fallback-concat⟩	33 955	500	465.85	99.25	98.92
MedDialog-EN	⟨fallback-concat⟩	479	479	132.34	30.71	30.23
MedQA-USMLE	`question+answer`	10 178	500	750.34	178.83	171.50

The "Field(s) used" column reports the actual provenance observed by the auditor: PubMed23n records carry a `text` field; Medical-Flashcards and MedDialog-EN expose only auxiliary fields that bypass both the priority list and the concatenation groups, so the loader's last-resort whole-record concatenation (⟨fallback-concat⟩) fires; MedQA-USMLE carries `question` and `answer`, picked up by the {`question`,`answer`} concatenation group. $\overline{\lvert T\rvert_{\text{char}}}$, $\overline{\lvert T\rvert_{\text{tok}}^{\text{Qwen}}}$ and $\overline{\lvert T\rvert_{\text{tok}}^{\text{Llama}}}$ are the mean character / Qwen-3.5 token / Llama-3.2 token lengths on the eval-capped subset. For the 20 PubMed23n shards we list a representative high/medium/low example and report the per-shard mean range across all 20 shards. According to the `seq_len_fit` audit, every eval document fits without truncation at every benchmarked context length (8K/12K/14K/16K) for both tokenizers.

8K / 12K / 14K / 16K context construction. For each target length $L \in \{8192, 12288, 14336, 16384\}$, the loader encodes one source document under `add_special_tokens=False` and packs one document per example. If the document is longer than $L+1$ tokens it is truncated; otherwise it is cyclically tiled until length $L+1$. The extra token is consumed by the shifted next-token loss, so the input fed to the model has length exactly L. The driver runs 40 batches at `batch_size`$=1$ per cell. Every eval document is far below 8K tokens (the longest Qwen-tokenized document is 964 tokens), so cyclic tiling, not truncation, is what produces every long-context input. This packing keeps the semantic content identical across 8K, 12K, 14K and 16K, so latency differences across lengths reflect only the cost of attention over a longer positional span.

B.4 Hardware and Software Stack

We list only the parts of the stack that affect the latency numbers in §5. Every reported timing is single-GPU, single-process, `batch_size`$= 1$.

Hardware. $1\times$ NVIDIA RTX 4090 D (24.6 GB GDDR6X, Ada Lovelace, compute capability 8.9, 114 SMs); Intel Core i9-14900K (32 threads). These experiments cover the self-built Transformer, Qwen-3.5-2B, and Llama-3.2-1B. The Qwen2.5-7B 32K experiment uses $1\times$ NVIDIA A800 80GB, 16 CPU cores, 80GB RAM, a 200GB system disk, and a 600GB data disk. All model runs use `bf16`; we report no `fp16` or quantized run.

Software. Ubuntu 24.04 on WSL2; CUDA 12.8 user-space (bundled by PyTorch); Python 3.12; `torch` 2.9.0, `triton` 3.5.0 (Tillet et al., 2019), `flash-attn` 2.8.1 (Shah et al., 2024); for Qwen-3.5 we additionally load `flash-linear-attention` 0.4.0 to cover the linear-attention layers (6 full + 18 linear out of 24, see Table 7); HuggingFace `transformers` 5.8.1.

Why these versions matter. The Qwen-3.5 16K bulge analyzed in §5.4 is a shared-memory event in FSA's score+top-K Triton kernel. The precise sequence length at which the spill triggers depends on the Triton release and on the CUDA runtime bundled with PyTorch, and may shift under other version pairs. The qualitative comparison with SCOPE is unaffected: SCOPE never enters a Triton kernel on the scoring path, and the sliding-window branch (`window_size` $= 512$) is identical across all four methods.

Relation to broader serving stacks. Our timing harness is single-process, with no batching, paged-KV, or prefill/decode disaggregation. SCOPE is orthogonal to serving systems such as vLLM (Kwon et al., 2023), SGLang (Zheng et al., 2024), DistServe (Zhong et al., 2024), Mooncake (Qin et al., 2024), and Hydragen (Juravsky et al., 2024), and would compose with each without modification.

B.5 Latency Measurement Protocol

All latencies use CUDA events with a single trailing `torch.cuda.synchronize()`. Two scripts share this core but differ in iteration budget. The reported CUDA-event measurements cover the stated attention/forward intervals; CPU chunking, lookup/refinement, host-to-device transfer, and cold-cache costs are not reported as separate wall-clock components.

Self-built Transformer. 10 untimed warm-up forwards, then 100 timed forwards under `torch.no_grad()`. Per-call event pairs are recorded without syncing between iterations, then read after one trailing sync. We report mean, std, and percentiles $\{p_{50}, p_{90}, p_{99}\}$ on the per-call vector. Every cell of Tables 1, 19, and 15 is the average of three independent process launches.

Commercial-backbone transfer. 5 untimed warm-up forwards, then 20 timed forwards per $(\text{mode}, L, \text{dataset})$ cell, with mode in $\{\text{Full}, \text{FSA}, \text{SCOPE}\}$. Inside the timed loop, an inner CUDA-event pair brackets only the attention path of every layer; an outer event pair brackets the whole forward. The driver runs 40 batches at `batch_size`$= 1$. Per-call samples are pooled across batches; Their median and P99 populate Figure 2 and Appendix Table 20. The outer pair gives the per-forward total used in Fig. 4.

What is excluded. Warm-up absorbs Triton JIT, cuBLAS handles, and first-touch caches. We use no CUDA Graph capture: it would freeze FSA's data-dependent control flow and erase the contrast we report in §5.4. We do not lock GPU clocks, so absolute millisecond values reflect default RTX 4090 D thermals. Relative FSA-vs-SCOPE numbers are unaffected.

B.6 Cross-Layer $\mathrm{topk_idx}$ Caching

The SCOPE *score+TopK* per-call latency of $\sim$0.19–0.20 ms relies on a simple invariant the main text leaves implicit: within a single forward pass, every transformer layer feeds an *identical* $\mathrm{topk_idx}$ tensor into the FSA sparse kernel. SCOPE therefore builds the tensor once and reuses it across the remaining $L-1$ layers.

Why the cache is bit-exact. By Algorithm 4 and the post-processing of §3.3, $\mathrm{topk_idx}$ depends only on $(\mathrm{cu_seqlens}, \mathbf{s}_{\mathrm{ref}}, B, K, n_{\mathrm{init}}, n_{\mathrm{local}}, H_{kv})$, none of which is a function of any per-layer hidden state. The first FSA layer populates the cache; layers $2, \ldots, L$ return the cached tensor directly. The cached and uncached results are bit-equal, so the cache is a pure latency optimization and does not enter the quality budget reported in §5.

B.7 Ablation Setup

The four perturbations of Table 15 are single-flag changes to the reference SCOPE path. We do not retrain weights, do not rebuild the chunk store, and keep the FSA hyperparameters fixed to the values in Table 7. Table 12 maps each row to the variable it patches.

All cells share the same budget: Qwen-3.5-2B at $L = 16{,}384$, batch size 1, 40 batches per (cell, dataset) pair, 5 warm-up and 20 timed iterations, giving 800 timed forward passes per cell-dataset pair. The four eval sources are the same as the rest of §5 (PubMed23n shards 0001–0020, Medical-Flashcards, MedDialog-EN, MedQA-USMLE), with the 500-record per-source cap of Appendix B.3. The only stochastic cell is B_2 (random miss-score draw); its 800-pass budget averages $\sim 8{\times}10^4$ score draws per dataset, which is enough for the tail-latency comparison reported in §5.5.

Table 12: Mapping between the ablation rows of Table 15 and the code path each cell patches. "Variable" is the field of the routing context; "CLI" is the flag passed to the benchmark script; "JSON" is the on-disk filename pattern.

Cell (Tab. 15)	**Code variable**	**Patch semantics**
Full SCOPE	*(none)*	reference path: store look-up → regressor on miss → two-level refinement
(A) no store, all → R	`ablation_no_store=True`	every chunk skips $\mathcal{H}$.LOOKUP and is forced through R
($\mathbf{B}_1$) miss → 0.5	`ablation_miss_strategy="constant"`; `ablation_constant_value=0.5`	every miss returns the constant 0.5 instead of $R(\mathbf{f}(C_i))$
($\mathbf{B}_2$) miss → rand	`ablation_miss_strategy="random"`	every miss returns `numpy.random.rand()` per chunk
(C) $\beta = \gamma = 0$	`ablation_no_refinement=True`	two-level refinement is bypassed: $\mathbf{s}_{\mathrm{ref}} \leftarrow \mathbf{s}_{\mathrm{prior}}$

Cell (Tab. 15)	**CLI flags added to base run**	**JSON tag**
Full SCOPE	*(none)*	`*__full`
(A)	`-ablation_no_store`	`*__A_no_store`
(B_1)	`-ablation_miss_strategy constant` `-ablation_constant_value 0.5`	`*__B_const`
(B_2)	`-ablation_miss_strategy random`	`*__B_random`
(C)	`-ablation_no_refinement`	`*__C_no_refine`

B.8 MedQA-USMLE Accuracy Scoring Protocol

The latency study reports a generation-fidelity bound $|\Delta\mathrm{PPL}| \leq 0.9$ between FSA and SCOPE. This subsection complements that bound with a decision-fidelity check on MedQA-USMLE (Jin et al., 2020): 10,178 four-way multiple-choice questions, each carrying a question, an options dictionary $(A/B/C/D)$, and a gold key. To stay within 24 GB on a single RTX 4090 D, the packed prompt length is fixed at $L = 3{,}072$ for both backbones.

Scoring rule. We use length-normalized conditional log-likelihood (LL-NORM), the de-facto zero-shot MCQ protocol of *lm-evaluation-harness* / HELM / BIG-bench and the original MedQA paper. For each option O_k, the score is

$$s(O_k \mid \mathcal{P}, Q) = \frac{1}{|\mathrm{tok}(O_k)|} \sum_{t=1}^{|\mathrm{tok}(O_k)|} \log P_\theta\left(o_k^{(t)} \mid \mathcal{P}, Q, \mathrm{TMPL}, o_k^{(<t)}\right),$$

with TMPL the literal string `"\n\nAnswer: "`, and $\mathcal{P}$ the in-domain prefix below. The predicted label is $\arg\max_k s(O_k \mid \mathcal{P}, Q)$. Length normalization is needed because options range from a single drug name to a full clinical sentence. The four per-option forward passes use the same attention configuration as the latency benchmark of §5.2.

Long-context packing. Each question is prefixed with a randomly drawn concatenation of PubMed23n passages until the tokenized prompt fills $L - 64$ tokens, leaving 64 tokens for the longest option. On overflow we left-truncate the prefix only, so Q and the option always sit at the right of the window. The prefix is shared across the four options, so SCOPE amortizes its store look-ups over all four scorings (cf. Appendix B.6).

Algorithm 2 MedQA-USMLE accuracy under FSA / SCOPE routing (length-normalized log-likelihood, $L = 3{,}072$).

Require: backbone $\theta \in \{$Qwen-3.5-2B, Llama-3.2-1B$\}$; tokenizer $\mathcal{T}$; mode $m \in \{$full, fsa_real, SCOPE_real$\}$; MedQA records $\mathcal{Q} = \{(Q_j, \mathcal{O}_j, k_j^\star)\}_{j=1}^{N}$; PubMed23n prefix pool $\mathcal{P}_{\text{pool}}$; target length $L = 3{,}072$; option budget $B_o = 64$; template TMPL = `"\n\nAnswer: "`; SCOPE components $(\mathcal{H}, R, \mathcal{C})$ if $m =$ SCOPE_real

Ensure: accuracy $\widehat{\text{Acc}}$; per-question records

if $m \neq$ full **then** inject FSA layers into θ
$n_{\text{correct}} \leftarrow 0$; $\mathcal{R} \leftarrow [\,]$
for $j = 1, \ldots, N$ **do**
// Build a ∼16K prefix shared by the four options
  $S_j \leftarrow$ TMPL $\|$ "Question: " $\|$ Q_j $\|$ TMPL
  $r_j \leftarrow L - B_o - |\mathcal{T}(S_j)|$; $\mathcal{P}_j \leftarrow [\,]$
  sample $\mathcal{P}_{\text{pool}}$ in random order; append passages to $\mathcal{P}_j$ until $|\mathcal{T}(\mathcal{P}_j)| \geq r_j$
  $\text{prompt}_j \leftarrow \mathcal{P}_j \,\|\, S_j$
  if $m =$ SCOPE_real **then**
    $\text{ctx}_j \leftarrow \textsc{BuildScopeContext}(\mathcal{H}, R, \mathcal{C}, \text{text} = \text{prompt}_j)$
    install ctx_j on FSA layers ▷ shared by all 4 options
  end if
// Score each option with length-normalized LL
  for $k \in \{A, B, C, D\}$ **do**
    $\mathbf{x} \leftarrow \mathcal{T}(\text{prompt}_j \,\|\, \text{“”} \,\|\, O_{j,k})$
    if $|\mathbf{x}| > L$ **then** left-truncate prefix only so that $|\mathbf{x}| = L$
    $\boldsymbol{\ell} \leftarrow \log \textsc{Softmax}\big(\theta_m(\mathbf{x}).\text{logits}\big)$ ▷ single forward; FSA / SCOPE routing applies
    let $n_k \leftarrow |\mathcal{T}(O_{j,k})|$
    $s_k \leftarrow \frac{1}{n_k} \sum_{t=L-n_k}^{L-1} \boldsymbol{\ell}_{t, \mathbf{x}_{t+1}}$ ▷ teacher-forced LL on option tokens
  end for
  $\hat{k}_j \leftarrow \arg\max_k \; s_k$; $\; y_j \leftarrow \mathbb{I}[\hat{k}_j = k_j^\star]$
  $n_{\text{correct}} \mathrel{+}= y_j$; $\mathcal{R}.\text{append}\big((j, \{s_k\}_k, \hat{k}_j, k_j^\star, y_j)\big)$
end for
return $\widehat{\text{Acc}} \leftarrow n_{\text{correct}}/N$, $\mathcal{R}$

Reporting. The accuracy $\widehat{\text{Acc}}$ produced by Algorithm 2 is reported per (backbone, mode) cell over the first 500 MedQA-USMLE records (matching the per-source 500-record cap from §B.3, so the question set is the same one whose median and P99 latency are reported in Tab. 20). The four (backbone, mode) cells are summarized in Table 13.

Table 13: MedQA-USMLE four-way MCQ accuracy on 500 questions. ΔAcc is SCOPE$-$FSA in absolute pp; ΔAcc/FSA is the relative change.

Backbone	Mode	Acc	ΔAcc	ΔAcc/FSA
Qwen-3.5-2B	FSA	0.2680	—	—
Qwen-3.5-2B	SCOPE	0.2660	-0.20 pp	-0.75%
Llama-3.2-1B	FSA	0.2280	—	—
Llama-3.2-1B	SCOPE	0.2180	-1.00 pp	-4.39%

Reporting. Algorithm 2 is run on the first 500 MedQA-USMLE records, matching the per-source cap used by the latency benchmark. The four cells appear in Table 13. The decision-fidelity claim of

§5.2 is operationalized as $|\widehat{\mathrm{Acc}}_{\mathrm{SCOPE}} - \widehat{\mathrm{Acc}}_{\mathrm{FSA}}| \leq 1$ pp on both backbones. With 500 questions, this gap sits well inside the binomial 95% CI of either backbone.

On the absolute level. The two FSA cells (26.80% and 22.80%) sit at the random-guess line of a four-way MCQ task. This is expected and is not a defect of FSA or SCOPE. Within the single-GPU budget of this study, both backbones are FSA-pretrained for only $1{,}000$ steps; the public chat checkpoints that usually report MedQA accuracy go through orders of magnitude more updates plus extensive SFT/RLHF on QA data. What this subsection measures is the decision-level delta between FSA and SCOPE at *matched* fine-tuning budget. The delta is ≤ 1 pp on both backbones and is consistent with the $|\Delta\mathrm{PPL}| \leq 0.9$ generation-fidelity bound. Pushing the absolute accuracy upward is orthogonal to the SCOPE contribution and is left to future work.

C Additional Results

This appendix collects additional evidence that supports §5. It includes cross-domain tests, a learned-routing baseline comparison, a sensitivity sweep, an incremental-update design, a timing-artifact check, and the raw numbers behind the main figures.

C.1 Transfer Quality Summary

Table 14 reports the full average PPL and loss deltas summarized in §5.2.

Table 14: Average ΔPPL / ΔLoss of SCOPE vs. FSA over the four medical corpora at 8K and 16K. Negative values indicate SCOPE improvement.

	$\bar{\Delta}$PPL		$\bar{\Delta}$Loss	
Model	8K	16K	8K	16K
Qwen-3.5	-1.2524	-0.8855	-0.0330	-0.0222
Llama-3.2	$+0.8510$	$+0.5689$	$+0.1941$	$+0.1366$

C.2 Full Ablation Results

Table 15 reports the full three-axis ablation summarized in §5.5.

Table 15: Three-axis ablation on Qwen-3.5 at 16K. **Full** is the reference SCOPE; the other rows switch off one path. AvgScr is the mean post-refinement chunk score. A1–B2 are the four medical corpora.

	Quality		**Diagnostic**		**Latency (mean)**		**P99 per dataset (ms)↓**			
Cell	ΔPPL	ΔLoss	Hit%	AvgScr	MedLat↓	P99↓	A1	A2	B1	B2
Full SCOPE	—	—	58.19	0.615	786.6	851.4	910.2	840.7	834.7	820.0
(A) no store, all $\to R$	$+0.032$	$+0.0007$	**0.00**	**0.989**	755.8	783.1	778.1	807.6	772.8	773.9
(B_1) miss $\to$ 0.5	$+0.021$	$+0.0005$	58.19	0.368	768.7	812.2	778.5	829.5	809.0	831.8
(B_2) miss $\to$ rand	$+0.016$	$+0.0004$	58.19	0.381	810.3	**882.7**	783.9	770.6	897.4	**1079.0**
(C) $\beta = \gamma = 0$	-0.001	$+0.0001$	58.19	0.567	**813.6**	852.4	775.7	890.2	888.1	855.6

C.3 Domain-Specific Store Boundary Case

At 8K, we also run domain-specific stores. The math store remains positive, with a 2.7% median-latency gain and a 96.6% hit ratio. The law store is the main boundary case. It has an 80.3% hit ratio, but SCOPE is 7.2% slower in median latency and 15.1% slower at P99. Its PPL still improves from 3.168 to 3.075. The issue is therefore inefficient block selection under domain structure shift, not a language-quality collapse.

Table 16: Domain-specific chunk-store details for the 8K boundary check.

Store/Test	Train docs	Chunks	FSA MedLat	SCOPE MedLat	Gain	Hit Ratio
MATH → MATH	7500	74213	430.1	418.4	+2.7%	96.6%
LexGLUE → Law	2000	47919	425.5	456.1	−7.2%	80.3%

C.4 Learned-Routing Baseline

We compare SCOPE with HashAttention using the same Llama-3.2-1B backbone and the same four medical datasets at 8K. SCOPE is faster in median latency on every dataset. The reduction ranges from 13.4% to 14.0% relative to HashAttention.

Table 17: Median-latency comparison with HashAttention at 8K on Llama-3.2-1B. Values are in milliseconds.

Dataset	HashAttention	SCOPE	SCOPE vs. HashAttention
PubMed23n	442.9	380.8	−14.0%
Medical Flashcards	443.2	383.8	−13.4%
MedDialog-EN	443.4	383.2	−13.6%
MedQA-USMLE	443.1	382.5	−13.7%

C.5 Sensitivity Sweep

We run a 55-run controlled sweep over chunk range, IDF weight, position weight, EMA rate, and ridge regularization. The original values are not always the best, but every tested setting keeps a positive median-latency gain. PPL stays in a narrow range from 1.8322 to 1.8366. This shows that SCOPE is not tied to one brittle hyperparameter point.

Table 18: Sensitivity sweep over SCOPE hyperparameters. "Paper" is the setting used in the main experiments.

Axis	Paper setting	Paper gain	Best raw setting	Robust reading
Chunk range	$(4, 5, 7)$	19.3%	$(4, 5, 6)$: 30.5%	$(3, 5, 7)$ gives 30.4% Med and 31.5% P99
IDF weight	0.6	21.0%	0.2: 26.9%	Lower IDF weight is better in this sweep
Position weight	0.4	15.5%	0.8: 26.4%	0.6 gives 24.4% with the best PPL
EMA α	0.2	19.5%	0.12: 25.5%	0.12–0.28 is a strong region
Ridge λ	10^{-3}	20.6%	10^{-8}: 28.7%	10^{-2}–1 gives 25.3–26.5%

C.6 Stationarity and Incremental Updates

SCOPE assumes that chunk importance is stable inside a deployment domain. One way to formalize this assumption is score drift. Let

$$\mathrm{score}_P(c) = \mathbb{E}_{x\sim P}[\phi(c, x)]$$

be the expected score of chunk c under domain distribution P. If $|\phi| \le B$, then

$$|\mathrm{score}_P(c) - \mathrm{score}_Q(c)| \le 2B\,\mathrm{TV}(P, Q).$$

Top-K routing remains stable when the score margin is larger than the drift term. In practice, SCOPE can monitor hit ratio and score margin. If either falls below a deployment threshold, the store can be updated or the system can fall back to the regressor or FSA path.

The store does not need to be rebuilt from scratch. Seen chunks can be updated by EMA. Unseen chunks can be inserted with regressor-estimated initial scores. The ridge predictor can be updated through sufficient statistics $A = X^\top X + \lambda I$ and $b = X^\top y$, without replaying the full corpus. LFU, LRU, or low-score eviction can bound memory. Our experiments evaluate the offline-store setting, but this design gives a practical path for drift-aware deployments.

C.7 Timing Artifact Around 6K

The small Llama-3.2+SCOPE bump around 6K tokens in the raw sweep is a measurement artifact. That partial sweep contains only two corpora, whereas the main grid averages four corpora. The bump is mainly caused by one PubMed run whose mean score+TopK time is affected by a rare timing spike. Its median is 0.0256 ms, which matches neighboring lengths, while its P99 reaches 0.851 ms. Since SCOPE's lookup path is microsecond-scale, occasional CUDA-event synchronization or Python scheduling noise can affect the mean on this small scale. With median aggregation, the bump disappears. We therefore do not interpret it as an algorithmic slowdown.

C.8 ChunkStore Hit Ratio and Top-K Select Ratio

Table 19 reports two diagnostics written by SCOPE on the self-built Transformer. The hit ratio is highest on the two corpora that overlap the offline ChunkStore most directly, MedQA-USMLE (65.12%) and PubMed23n (57.35%). It is lowest on the two cross-format corpora, Medical-Flashcards (46.56%) and MedDialog-EN (47.24%), where dialogue and flashcard formats differ from the long-paragraph chunks the store ingested. The ~ 18.6 pp spread therefore reflects coverage of the offline prior, not per-corpus tuning. The Select Ratio sits in a tight 34.97–43.93% band, matching the fixed `topk_ratio`$= 0.4$ of Table 7. The ± 5 pp residual is the rounding artifact of $\lceil$ `topk_ratio` $\cdot\, n_{\text{chunk}} \rceil$ on short sequences. The Select Ratio is therefore a guardrail showing that the target sparsity floor is preserved on every corpus, even the two low-hit settings where the regressor carries the load.

Table 19: ChunkStore hit ratio and Top-K block select ratio of SCOPE on the four medical corpora (self-built 10-layer Transformer).

Dataset	Hit Ratio	Select Ratio
PubMed23n	57.35%	39.94%
Medical-Flashcards	46.56%	34.97%
MedDialog-EN	47.24%	43.93%
MedQA-USMLE	65.12%	39.40%

C.9 Per-Operator Latency Decomposition

Figure 4 decomposes the per-forward latency of the self-built Transformer into two operators (`score_topk` and `attn_layer`) at two granularities (per call, per forward), separately for the four medical corpora and the four attention architectures. Two readings stand out. First, the selection operator drops from 40.4 ms per call on FSA and 141.6 ms on NSA to a corpus-stable 0.189–0.196 ms on SCOPE, with per-corpus spread below 0.01 ms. The $\mathcal{O}(1)$ store look-up plus constant-cost refinement is therefore data-independent in practice as well as in theory. Second, the dense attention operator itself moves only modestly (SCOPE 115.6–117.5 ms vs. FSA 152.4–152.9 ms vs. Full 175.0–175.1 ms). The ~ 350 ms per-forward MedLat reduction reported in Table 1 is therefore explained by the shrinkage of the score+TopK stage, not by a faster attention kernel.

C.10 Raw Per-Cell Numbers Behind Figures 2–4

This subsection tabulates every cell that drives Figures 2, 3, and 4, so that any figure cell can be read off without re-extracting the underlying JSON. The aggregation protocol is the one in Appendix B.5. Each table includes the diagnostic fields that the corresponding figure does not display (PPL/ΔPPL for the transfer track; hit ratio, select ratio, and average chunk score for the self-built track; per-call vs. per-forward ratios for the score+TopK and attention operators).

C.10.1 Per-Cell Latency and Quality Behind Figure 2

Table 20 reproduces the $4{\times}4{\times}2{=}32$ cells of Figure 2 together with the per-cell PPL. The `Red.` columns are $(t_{\text{FSA}} - t_{\text{SCOPE}})/t_{\text{FSA}}$, and ΔPPL is $\text{PPL}_{\text{SCOPE}} - \text{PPL}_{\text{FSA}}$. The latency reduction is positive on every Llama-3.2-1B cell. On Qwen-3.5-2B, the only negative cell is (12K, Med-Flashcards), where SCOPE is -0.38% slower at the median (-0.87% at P99).

Table 20: Per-cell raw latency and quality behind Figure 2. MedLat / P99 in ms; Red. is $(t_{\text{FSA}} - t_{\text{SCOPE}})/t_{\text{FSA}}$; ΔPPL is $\text{PPL}_{\text{SCOPE}} - \text{PPL}_{\text{FSA}}$ (negative favors SCOPE).

		MedLat (ms)			P99 (ms)			PPL		
L	Dataset	FSA	SCOPE	Red.	FSA	SCOPE	Red.	FSA	SCOPE	ΔPPL
Qwen-3.5-2B										
8K	PubMed23n	419.91	394.40	+6.07%	438.53	403.97	+7.88%	65.61	63.23	−2.375
8K	Med-Flashcards	419.42	397.83	+5.15%	438.39	407.49	+7.05%	20.68	20.33	−0.350
8K	MedDialog-EN	423.33	394.25	+6.87%	463.22	403.12	+12.97%	22.56	21.88	−0.688
8K	MedQA-USMLE	421.29	411.70	+2.28%	431.18	428.69	+0.58%	34.82	33.22	−1.597
12K	PubMed23n	610.52	561.45	+8.04%	625.54	572.90	+8.41%	65.10	63.15	−1.945
12K	Med-Flashcards	589.55	591.79	−0.38%	617.89	623.23	−0.87%	20.83	20.69	−0.138
12K	MedDialog-EN	607.50	589.19	+3.01%	628.88	597.92	+4.92%	23.39	22.63	−0.763
12K	MedQA-USMLE	606.17	588.89	+2.85%	626.26	604.97	+3.40%	35.01	33.83	−1.180
14K	PubMed23n	721.63	633.62	+12.20%	771.64	657.33	+14.81%	65.08	63.25	−1.836
14K	Med-Flashcards	731.32	673.50	+7.91%	744.90	689.84	+7.39%	20.89	20.84	−0.044
14K	MedDialog-EN	722.35	674.19	+6.67%	752.11	684.99	+8.92%	23.81	23.02	−0.785
14K	MedQA-USMLE	725.66	673.19	+7.23%	746.28	695.68	+6.78%	35.22	34.14	−1.077
16K	PubMed23n	913.62	745.43	+18.41%	1153.17	754.85	+34.54%	65.16	63.40	−1.760
16K	Med-Flashcards	900.95	777.25	+13.73%	1074.46	787.92	+26.67%	20.97	20.99	+0.019
16K	MedDialog-EN	845.74	775.23	+8.34%	1092.75	811.69	+25.72%	24.25	23.44	−0.807
16K	MedQA-USMLE	961.47	774.54	+19.44%	1141.25	793.30	+30.49%	35.47	34.48	−0.994
Llama-3.2-1B										
8K	PubMed23n	422.89	380.81	+9.95%	544.95	478.49	+12.20%	5.22	6.49	+1.276
8K	Med-Flashcards	425.50	383.82	+9.79%	547.78	524.06	+4.33%	1.72	1.71	−0.008
8K	MedDialog-EN	423.19	383.23	+9.44%	556.94	499.07	+10.39%	1.95	1.98	+0.034
8K	MedQA-USMLE	427.66	382.54	+10.55%	554.32	505.95	+8.73%	2.90	5.00	+2.103
12K	PubMed23n	554.89	481.08	+13.30%	669.41	595.53	+11.04%	5.42	6.23	+0.816
12K	Med-Flashcards	557.60	479.20	+14.06%	691.38	600.14	+13.20%	1.72	1.69	−0.028
12K	MedDialog-EN	557.07	475.13	+14.71%	677.43	595.93	+12.03%	1.96	1.96	+0.008
12K	MedQA-USMLE	557.77	476.48	+14.57%	672.78	597.85	+11.14%	2.91	4.91	+2.004
14K	PubMed23n	631.25	535.62	+15.15%	765.35	643.96	+15.86%	5.51	6.12	+0.611
14K	Med-Flashcards	627.94	529.83	+15.62%	755.15	652.05	+13.65%	1.73	1.69	−0.037
14K	MedDialog-EN	635.96	528.34	+16.92%	746.61	648.30	+13.17%	1.96	1.96	−0.002
14K	MedQA-USMLE	631.75	530.91	+15.96%	742.52	659.66	+11.16%	2.92	4.90	+1.974
16K	PubMed23n	727.35	578.15	+20.51%	843.08	695.09	+17.55%	5.58	5.96	+0.382
16K	Med-Flashcards	724.74	582.96	+19.56%	839.82	687.89	+18.09%	1.73	1.68	−0.045
16K	MedDialog-EN	725.12	577.00	+20.43%	834.53	685.56	+17.85%	1.96	1.96	−0.009
16K	MedQA-USMLE	720.31	576.52	+19.96%	832.74	675.22	+18.92%	2.92	4.87	+1.948

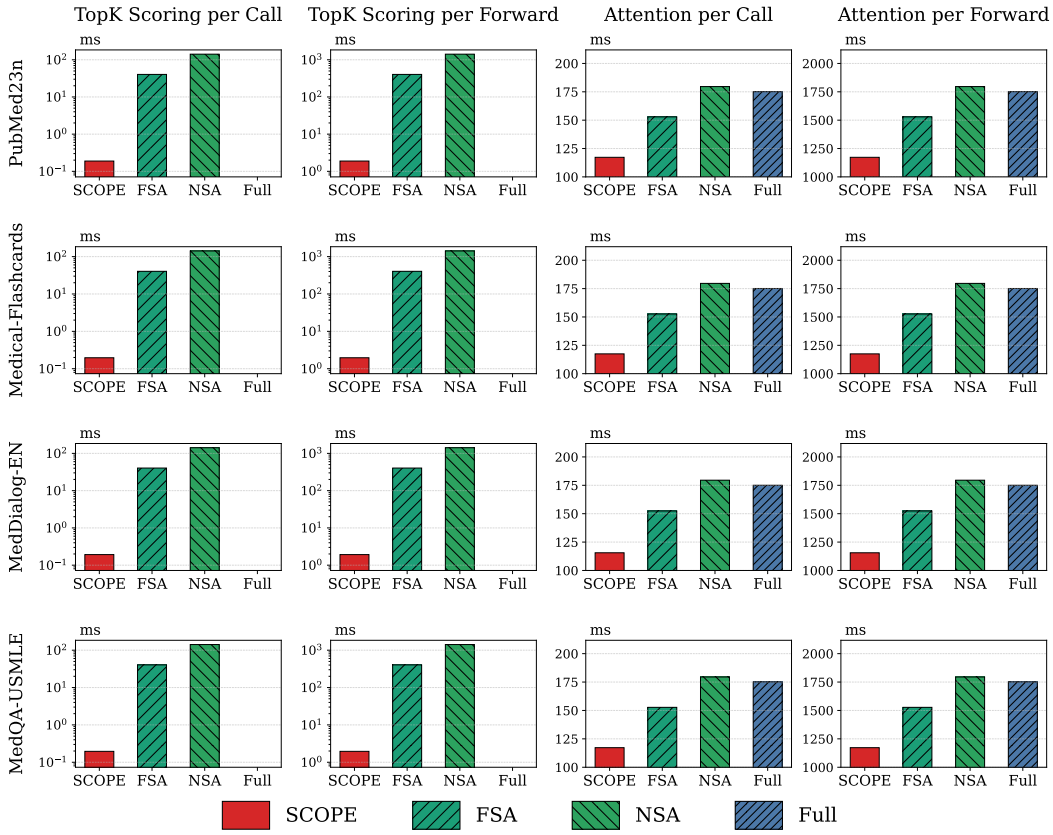


Figure 4: Per-kernel latency breakdown across four medical corpora (rows) and four micro-benchmarks (columns) for SCOPE, FSA, NSA, and Full Attention. Lower is better; bars are log-scaled where applicable.

C.10.2 Per-Length Latency Sweep Behind Figure 3

Table 21 aggregates, for every measured context length and every backbone, the four-dataset average of median/P99 forward latency, the average score+TopK per-forward time, and the corresponding latency reduction. The "n" column reports how many of the four medical datasets were measured at each length; rows with $n<4$ are partial sweeps. The $\mathrm{Spd}_{\mathrm{score}}$ column is the per-forward score+TopK speedup; it grows from $\sim 400\times$ at 4K to ~ 5000–$8000\times$ at 16K on both backbones, because the FSA dense scoring kernel scales while the SCOPE look-up does not. The Qwen 18K row carries the silent OOR fallback discussed in §5.4: FSA quietly drops to the `init+local` block path while SCOPE still runs the full sparse branch, which produces the negative reduction. We keep the row for completeness.

Table 21: Per-length latency sweep behind Figure 3. Each row averages medians or P99s over the datasets available at that length. n is the dataset count; rows with $n<4$ are partial sweeps. $\text{Score}_{\text{fwd}}$ is the per-forward score+TopK time; the rightmost column is its FSA/SCOPE ratio.

		MedLat (ms)			P99 (ms)			Score+TopK per-fwd (ms)		
L	n	FSA	SCOPE	Red.	FSA	SCOPE	Red.	FSA	SCOPE	$\text{Spd}_{\text{score}}$
Qwen-3.5-2B										
4K	4	215.43	201.14	+6.64%	233.66	215.79	+7.65%	4.46	0.0105	∼425×
6K	2	311.69	291.33	+6.53%	324.89	318.51	+1.96%	4.14	0.0098	∼425×
8K	4	420.99	399.55	+5.09%	442.83	410.82	+7.23%	5.88	0.0102	∼580×
10K	2	510.28	469.63	+7.97%	526.73	484.63	+7.99%	7.18	0.0101	∼709×
12K	4	603.43	582.83	+3.41%	624.64	599.76	+3.98%	9.10	0.0098	∼929×
13K	4	667.61	617.97	+7.43%	694.59	635.01	+8.58%	35.75	0.0100	∼3561×
14K	4	725.24	663.62	+8.50%	753.73	681.96	+9.52%	35.73	0.0099	∼3594×
15K	4	791.44	718.87	+9.17%	917.72	743.26	+19.01%	49.26	0.0099	∼4968×
16K	4	905.45	768.11	+15.17%	1115.41	786.94	+29.45%	79.44	0.0097	∼8192×
16.7K	4	888.74	795.12	+10.53%	1073.42	822.14	+23.41%	69.48	0.0104	∼6653×
17K	4	910.50	826.18	+9.26%	1044.59	861.31	+17.55%	59.17	0.0101	∼5846×
17.5K	1	1022.67	900.89	+11.91%	1330.69	1041.82	+21.71%	54.70	0.0113	∼4852×
18K⋆	1	1249.66	2149.96	−72.04%	1969.99	2538.33	−28.85%	62.92	0.0102	∼6177×
Llama-3.2-1B										
4K	4	305.14	292.83	+4.04%	438.75	413.81	+5.69%	13.07	0.0301	∼435×
6K	2	353.98	332.55	+6.05%	482.61	447.13	+7.35%	20.68	0.0462	∼448×
8K	4	424.81	382.60	+9.94%	551.00	501.89	+8.91%	38.53	0.0273	∼1409×
10K	2	481.66	427.92	+11.16%	602.24	542.81	+9.87%	49.69	0.0261	∼1904×
12K	4	556.83	477.97	+14.16%	677.75	597.36	+11.86%	73.86	0.0264	∼2798×
13K	4	588.97	502.44	+14.69%	694.67	608.13	+12.46%	81.12	0.0262	∼3096×
14K	4	631.73	531.17	+15.92%	752.41	650.99	+13.48%	97.73	0.0267	∼3665×
15K	4	659.13	551.01	+16.40%	770.68	663.13	+13.96%	101.09	0.0264	∼3833×
16K	4	724.38	578.66	+20.12%	837.54	685.94	+18.10%	135.25	0.0267	∼5071×
16.7K	4	759.43	587.14	+22.69%	852.35	708.94	+16.83%	138.00	0.0283	∼4884×
17K	4	764.08	590.41	+22.73%	855.90	697.64	+18.49%	143.08	0.0290	∼4927×
18K	4	813.69	621.83	+23.58%	894.36	725.43	+18.89%	158.02	0.0266	∼5937×

⋆ The Qwen 18K row is the single-dataset A1_pubmed23n sample on which FSA's `CompressedAttention.apply` hits the SM shared-memory limit and silently falls back to the `init+local` block path (§5.4). FSA's reported latency therefore drops abruptly while SCOPE continues to execute the full sparse-selection branch, which produces the negative reduction. Both rows are reported as is, and the missing three datasets are not back-filled.

C.10.3 Per-Kernel Decomposition Behind Figure 4 (Self-built 10-Layer Transformer)

Table 22 reports the 16 cells behind Figure 4 on the self-built Transformer. Each row records four micro-benchmark columns (score+TopK per call and per forward, attention per call and per forward), end-to-end MedLat and P99, mean PPL and Loss, and the SCOPE-only diagnostics Hit, Sel, and AvgScr. All numbers are three-run aggregates. Across the four corpora, SCOPE reduces the per-call score+TopK cost from about 40 ms in FSA and 141 ms in NSA down to about 0.19 ms, and keeps per-call attention within about 35 ms of FSA. This is the same attribution that Figure 4 shows in log scale.

Table 22: Per-kernel decomposition behind Figure 4 (self-built 10-layer Transformer at $L = 49152$). The four kernel columns are score+TopK and attention, each per call and per forward (ms). End-to-end MedLat/P99 and PPL/Loss match Table 1. Hit / Sel / AvgScr are SCOPE-only.

		Score+TopK (ms)		Attention (ms)		End-to-end (ms)		Quality		SCOPE-only		
Corpus	Method	/call	/fwd	/call	/fwd	MedLat	P99	PPL	Loss	Hit%	Sel%	AvgScr
PubMed23n	Full Attn	—	—	175.10	1750.98	2432.37	2442.40	678.37	6.5197	—	—	—
PubMed23n	NSA	141.672	1416.724	179.60	1795.99	2474.76	2483.19	579.15	6.3616	—	—	—
PubMed23n	FSA	40.422	404.217	152.91	1529.09	2203.82	2227.06	392.62	5.9728	—	—	—
PubMed23n	SCOPE	0.189	1.890	117.26	1172.60	1852.43	1881.64	392.78	5.9733	57.35	39.49	0.6081
Med-Flashcards	Full Attn	—	—	175.12	1751.18	2433.28	2437.50	1084.49	6.9889	—	—	—
Med-Flashcards	NSA	141.606	1416.055	179.61	1796.10	2473.87	2490.31	960.87	6.8678	—	—	—
Med-Flashcards	FSA	40.429	404.291	152.62	1526.17	2200.88	2223.19	605.99	6.4069	—	—	—
Med-Flashcards	SCOPE	0.196	1.964	117.46	1174.61	1855.47	1886.28	606.89	6.4083	46.56	34.97	0.6829
MedDialog-EN	Full Attn	—	—	175.04	1750.43	2432.43	2436.28	1596.23	7.3754	—	—	—
MedDialog-EN	NSA	141.623	1416.229	179.51	1795.07	2474.00	2479.62	1273.02	7.1492	—	—	—
MedDialog-EN	FSA	40.428	404.281	152.41	1524.07	2200.76	2220.87	952.61	6.8592	—	—	—
MedDialog-EN	SCOPE	0.192	1.924	115.58	1155.77	1828.20	1893.39	952.08	6.8587	47.24	43.93	0.6250
MedQA-USMLE	Full Attn	—	—	175.13	1751.28	2433.48	2437.29	600.73	6.3981	—	—	—
MedQA-USMLE	NSA	141.595	1415.946	179.46	1794.56	2471.77	2504.25	509.64	6.2337	—	—	—
MedQA-USMLE	FSA	40.439	404.393	152.74	1527.42	2203.19	2223.39	374.06	5.9244	—	—	—
MedQA-USMLE	SCOPE	0.196	1.959	117.24	1172.35	1857.23	1882.76	374.42	5.9254	65.12	39.40	0.5680

D Extended Content Details

This appendix expands four parts of the main text. The first three subsections support §3: the offline build pipeline, the inference pseudocode, and the padding and broadcast convention used by the FSA kernel. The fourth subsection extends §2 with a wider related-work positioning. The last subsection collects the result discussions referenced from §5.

D.1 Build Pipeline Details

This subsection expands the four offline stages summarized in §3.2. All steps run on the CPU.

Corpus extraction. The builder reads the domain dataset from local storage. Line-format files are parsed one record at a time, and column-format files are loaded in fixed-size batches. Main text fields are picked by a fixed priority schedule (Appendix B.2). When a sample matches a known concatenation group, the relevant fields are joined to keep context. Two truncation steps and a per-file document cap stop any long document or any single source from dominating the store.

Semantic chunking. The chunker uses four design choices. First, the basic unit is the word boundary recovered from a subword tokenizer, with target length $\ell \in [4, 7]$. This unit is finer than a sentence and coarser than a word. Second, a symbolic pattern routes punctuation, dashes, quotes, parentheses, and bare numbers into seven buckets, which keeps low-information keys out of the store. Third, an idempotent normalizer ϕ applies compatibility normalization, dash unification, whitespace compression, lower-casing, and digit zeroing. Build and inference must use the same ϕ. Fourth, once the current word count reaches $\ell^\star = 5$, a soft cut is allowed only at stop words. The chunker also returns the token index set $\mathcal{I}_i$ of each chunk for the granularity bridge.

Algorithm 3 SCOPE Chunk-Store Construction (with inlined heuristic scorer).

Require: corpus $\mathcal{F}$; tokenizer $\mathcal{T}$; chunker $\mathcal{C}$; EMA rate $\alpha = 0.2$; store cap M; ridge $\lambda = 10^{-3}$
Ensure: chunk store $\mathcal{H}$; regressor R

$\mathcal{H}, \Phi \leftarrow \emptyset, \emptyset$; $N \leftarrow 0$
// Pass 1: chunk-level document frequency
for $T \in \mathcal{F}$ **do**
  truncate T; $(\mathbf{C}, \cdot) \leftarrow \mathcal{C}.\textsc{chunkText}(T)$
  for all $c \in \textsc{set}(\mathbf{C})$ **do** $\Phi[c] \mathrel{+}= 1$
  end for
  $N \mathrel{+}= 1$
end for
$\mathrm{IDF}[c] \leftarrow \log \frac{N+1}{\Phi[c]+1} + 1 \;\forall\, c$
// Pass 2: heuristic score + EMA write
for $T \in \mathcal{F}$ **do**
  $(\mathbf{C}, \cdot) \leftarrow \mathcal{C}.\textsc{chunkText}(T)$; $n \leftarrow |\mathbf{C}|$
  for $i = 0, \ldots, n-1$ **do**
    if C_i is a symbol bucket **then** $s \leftarrow 0.1$
    else
      $\iota \leftarrow \min(\mathrm{IDF}[C_i]/6,\ 1)$
      $\pi \leftarrow \max\big(0.5,\ 1 - 0.5\, i/n\big)$ ▷ $n = 1 \Rightarrow \pi = 1$
      $s \leftarrow \textsc{Clip}(0.6\,\iota + 0.4\,\pi,\ 0,\ 1)$
    end if
    if $C_i \in \mathrm{dom}(\mathcal{H})$ **then**
      $\mathcal{H}[C_i].\mathrm{score} \leftarrow \alpha s + (1-\alpha)\mathcal{H}[C_i].\mathrm{score}$
      $\mathcal{H}[C_i].\mathrm{count} \mathrel{+}= 1$
    else if $|\mathcal{H}| < M$ **then** $\mathcal{H}[C_i] \leftarrow (s, 1)$
    end if
  end for
end for
// Regression predictor
$(\tilde{\mathbf{C}}, \tilde{\mathbf{s}}) \leftarrow \textsc{Top}\ 50\mathrm{K}(\mathcal{H},\ \text{by count})$
$\mathbf{X} \leftarrow [\,\mathbf{f}(\tilde{C}_j)\,;\,1\,]_j$; $\mathbf{w}^\star \leftarrow (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\tilde{\mathbf{s}}$; $R \leftarrow \mathbf{w}^\star$
return $(\mathcal{H},\ R)$

Two-pass scanning. Pass 1 builds the IDF table on chunk-level document frequency. The frequency counter is then released. Pass 2 reuses the same chunking pipeline and writes a heuristic score under the EMA insertion rule of §3.2. The hit count κ records how often each chunk has been seen and serves as supervision weight for the regressor.

Regression predictor details. Supervision comes from the EMA scores of the most-hit chunks, sorted by κ. Symbol buckets are filtered before training. The rare-word threshold is $\tau_{\text{idf}} = 3.0$. A gradient-boosting variant is available but is not used in our experiments.

Algorithm 3 summarizes the offline build with the heuristic scorer inlined into Pass 2. The symbol-bucket short-circuit, the IDF saturation, and the position prior collapse into a four-line subroutine. The trailing ridge solve produces the regressor R.

D.2 Inference Pseudocode

Algorithm 4 merges prior collection and the two-level refinement into one pass over the chunk sequence. Stage 1 unifies the store lookup and the regressor fallback. Stage 2 applies the intra-chunk additive correction. Stage 3 adds the bidirectional neighbor bonus and the isolation penalty. The isolation indicator is $\mathbb{I}_{\text{iso}}^{(i)} = 1$ iff $s_{\text{adj}}^{(i)} \geq \tau_h$ and both $s_{\text{adj}}^{(i\pm1)} < \tau_h$.

D.3 Padding and Broadcast

The FSA kernel requires $\mathrm{topk_idx}$ to have a fixed length K along the last dimension. When the number of valid blocks n_{avail} is less than K, the missing slots are padded with -1, and the kernel

Algorithm 4 SCOPE Prior Score Collection and Two-Level Refinement.

Require: chunks $\{C_i\}_{i=1}^{n}$; store $\mathcal{H}$; regressor R; $\beta_{1:3}=(0.1, 0.15, 0.05)$; $\gamma=0.08$; $\tau_h=0.6$; $\ell^\star=5$
Ensure: refined scores $\mathbf{s}_{\text{ref}} \in [0,1]^n$; hit count κ

$\kappa \leftarrow 0$
// Stage 1: prior collection (hit / miss unified)
for $i=0,\dots,n-1$ **do**
  $s \leftarrow \mathcal{H}.\textsc{lookup}(C_i)$
  if $s=\textsc{None}$ **then** $s \leftarrow \textsc{Clip}(R.\textsc{predict}(C_i), 0, 1)$
  else $\kappa \mathrel{+}= 1$
  end if
  $s^{(i)}_{\text{prior}} \leftarrow s$
end for
// Stage 2: intra-chunk additive correction
for $i=0,\dots,n-1$ **do**
  $s^{(i)}_{\text{adj}} \leftarrow \textsc{Clip}\big(s^{(i)}_{\text{prior}} + \beta_1 f^{(i)}_{\text{rare}} + \beta_2 f^{(i)}_{\text{dom}} - \beta_3 \sigma(|C_i| - \ell^\star),\ 0,\ 1\big)$
end for
// Stage 3: inter-chunk neighbour reward / isolation penalty
for $i=0,\dots,n-1$ **do**
  $\Delta \leftarrow 0$
  if $i>0 \ \wedge\ s^{(i-1)}_{\text{adj}} \geq \tau_h$ **then** $\Delta \mathrel{+}= \gamma$
  end if
  if $i<n-1 \ \wedge\ s^{(i+1)}_{\text{adj}} \geq \tau_h$ **then** $\Delta \mathrel{+}= \gamma$
  end if
  if $s^{(i)}_{\text{adj}} \geq \tau_h$ and both neighbours $< \tau_h$ **then** $\Delta \mathrel{-}= 0.1$
  end if
  $s^{(i)}_{\text{ref}} \leftarrow \textsc{Clip}(s^{(i)}_{\text{adj}} + \Delta, 0, 1)$
end for
return $(\mathbf{s}_{\text{ref}}, \kappa)$

skips any slot equal to -1:

$$\widetilde{\mathcal{T}}[k] = \begin{cases} \mathcal{T}_{\text{ord}}[k], & 0 \le k < n_{\text{avail}}, \\ -1, & n_{\text{avail}} \le k < K. \end{cases}$$

SCOPE then broadcasts $\widetilde{\mathcal{T}}$ along the head dimension H_{kv} and the query dimension L, so all KV heads share the same top-K set.

D.4 Extended Related Work

This subsection gives the broader related-work positioning that §2 points to. None of these methods are directly comparable to SCOPE, but they bound the Pareto frontier on the long-context pipeline. We group them into two clusters.

Orthogonal scoring methods. MoBA (Lu et al., 2025a) uses a parameter-free MoE-style router. Its authors note that the router lacks long-context detail, and it is also an online analogue of SCOPE's offline score. XAttention (Xu et al., 2025) is a prefill-only antidiagonal scorer for dense backbones. Quest (Tang et al., 2024) is the canonical query-aware page-level selector and reports query-dependent criticality flips and page-boundary fragmentation. SCOPE avoids both effects by aligning with NSA's $\ell' = 64$ blocks. MInference (Jiang et al., 2024) and FlexPrefill (Lai et al., 2025) keep per-head offline pattern libraries with online switching. Both cover only prefill, and both require training. KVLink (Yang et al., 2025a) reuses KV across requests on a different axis and can be combined with SCOPE.

2025 sparse-attention and KV-cache lines. A second wave of 2025 work targets neighboring parts of the long-context pipeline, and all of these methods are composable with SCOPE. On the *sparse-attention* axis, LServe (Yang et al., 2025c) and SampleAttention (Zhu et al., 2025) unify or adapt block-sparse patterns at serving time. Star Attention (Acharya et al., 2025) distributes long sequences blockwise. SeerAttention-R (Gao et al., 2025a) extends gate distillation to long reasoning. Rectified (Sun et al., 2025), Delta (Willette et al., 2025), and Hierarchical Semi-Structured (Wang & Wang, 2026) variants further refine the score-correction trade-off. SamKV (Cao et al., 2026) extends sparse selection to multi-context KV caches in RAG. On the *KV-cache* axis, RocketKV (Behnam

et al., 2025) chains two compression stages. SnapKV (Li et al., 2024a), PyramidKV (Cai et al., 2025), Ada-KV (Feng et al., 2025), and SpindleKV (Tang et al., 2025) target eviction strategies. SCOPE is independent of all of them. It removes the scoring kernel rather than reordering it, and it leaves the surviving sparse branches, the sliding-window branch, and any KV compression unchanged.

Orthogonal composable lines. Two further lines target points that SCOPE does not touch. The first is retrieval-augmented generation. Methods such as LongRAG (Zhao et al., 2024), LongRefiner (Jin et al., 2025), HiChunk (Lu et al., 2025b), MacRAG (Lim et al., 2025), Adaptive-k (Taguchi et al., 2025), UncertaintyRAG (Li et al., 2024b), and Self-RAG (Asai et al., 2024) act at the retrieval layer. They shrink or rerank the context before it reaches the model. SCOPE keeps the full context resident on the GPU and only changes how each chunk is scored. Any RAG front-end can therefore feed SCOPE a shorter context, and the two stages compose without conflict. The second line is speculative decoding. AdaSpec (Huang et al., 2025), DSDE (Yang et al., 2025b), and the analysis in Yan et al. (2025) accelerate the decoding half. SCOPE accelerates the prefill half. The two approaches could be complementary, but their combined speedup remains to be measured.

Long-context benchmarks left to future work. Our evaluation uses a medical corpus testbed and checks quality parity through two signals: $|\Delta\mathrm{PPL}| \leq 0.9$ on average and a MedQA-USMLE decision-fidelity check. Standard long-context benchmarks such as RULER (Hsieh et al., 2024), LongBench v2 (Bai et al., 2025), and BABILong (Kuratov et al., 2024) stress needle-in-a-haystack recall in more controlled settings. A drop-in scoring replacement like SCOPE should preserve these signals. Full RULER and LongBench v2 sweeps are left to future work.

D.5 Extended Results Discussion

This subsection expands four points referenced from §5 and the Conclusion: the tight P99–MedLat gap on the self-built Transformer, the per-operator detail behind the end-to-end speedup, the per-cell ablation discussion, and the attribution that supports the Conclusion.

P99–MedLat compression and quality match. On the self-built Transformer, the relative gain in P99 ranges from 14.75% to 15.51%. The gap from the median gain is at most 0.5 pp. SCOPE therefore compresses the whole latency distribution, not only the peak. The ChunkStore path is data-independent and deterministic. It has no scoring kernel that spills to global memory when the on-chip workspace overflows, and no long-tail spikes. On MedDialog-EN, SCOPE is 0.53 PPL lower than FSA. The offline priors are trained on a large corpus and stay globally stable after refinement, which makes them less sensitive to unusual local correlations in any single input.

Per-operator detail (Figure 4). SCOPE is about $213\times$ faster than FSA per call on the score+TopK kernel and about $745\times$ faster than NSA. The fluctuation of SCOPE across the four corpora is below 0.01 ms. The fluctuation for FSA and NSA is below 0.05 ms. The selection cost is therefore determined by control flow, not by data. This confirms the "$\mathcal{O}(1)$ hash lookup plus constant-cost two-level refinement" claim in §3. In the per-forward column, the values are uniformly $10\times$ the per-call values, which matches a 10-layer model. The gap between SCOPE (~ 2 ms) and FSA (~ 404 ms) is about 402 ms. This independently explains the end-to-end MedLat reduction of about 350 ms in Table 1. The roughly 402 ms score+TopK saving is partly offset by 34-51 ms higher non-scoring attention time under SCOPE.

Per-cell ablation discussion (Table 15). When the ChunkStore is disabled, every chunk goes through the regressor. AvgScr saturates from 0.615 to 0.989, the hit ratio drops to zero, and the score distribution loses its discriminative power. When the regressor is replaced by a constant or random score, AvgScr drops to 0.368 and 0.381. The real cost shows up in tail latency: on MedQA, B_2 raises P99 from 820 ms to 1079 ms; on MedDialog, P99 rises from 835 ms to 897 ms. When the β/γ refinement is disabled, PPL is almost unchanged ($\Delta\mathrm{PPL} = -0.001$), but MedLat rises from 786.6 ms to 813.6 ms. Refinement therefore acts on median latency, not on quality. It steers selection toward consecutive high-score chunks, which preserves the local regularity of KV block accesses.

Quantitative attribution supporting the Conclusion. The MedLat reduction on the self-built Transformer averages 16.07% across the four corpora, with a standard deviation of 0.50 pp. The

operator-level breakdown attributes the gain to the selection operator. The per-forward selection gap is about 402 ms. The end-to-end MedLat reduction is about 350 ms. The two numbers match within tens of milliseconds, and the small attention drop closes the remaining gap. On released LLM backbones, the speedup grows almost monotonically with context length until saturation. The shape of the curves in Figure 3, together with the inflection in the right panel, points to on-chip workspace overflow in FSA at long contexts. The three components of SCOPE act on three separate axes: the ChunkStore on score distribution, the regressor on tail latency, and the two-level refinement on median latency. They are therefore not redundant.