Stream: Scaling up Mechanistic Interpretability to Long Context in LLMs via Sparse Attention

J Rosser[*]
University of Oxford
jrosser@robots.ox.ac.uk

José Luis Redondo García
Spotify
Spain

Gustavo Penha
Spotify
Netherlands

Konstantina Palla
Spotify
UK

Hugues Bouchard
Spotify
Spain

Abstract

As Large Language Models (LLMs) scale to million-token contexts, traditional Mechanistic Interpretability techniques for analyzing attention scale quadratically with context length, demanding terabytes of memory beyond 100,000 tokens. We introduce SPARSE TRACING, a novel technique that leverages dynamic sparse attention to efficiently analyze long context attention patterns. We present STREAM, a compilable hierarchical pruning algorithm that estimates per-head sparse attention masks in near-linear time $O(T \log T)$ and linear space $O(T)$, enabling one-pass interpretability at scale. STREAM performs a binary-search-style refinement to retain only the top-k key blocks per query while preserving the model's next-token behavior. We apply STREAM to long chain-of-thought reasoning traces and identify thought anchors while pruning 97-99% of token interactions. On the RULER benchmark, STREAM preserves critical retrieval paths while discarding 90-96% of interactions and exposes layer-wise routes from the needle to output. Our method offers a practical drop-in tool for analyzing attention patterns and tracing information flow without terabytes of caches. By making long context interpretability feasible on consumer GPUs, SPARSE TRACING helps democratize chain-of-thought monitoring. Code is available at https://github.com/spotify-research/stream-mechinterp/.

1 Introduction

Extending the context length of Large Language Models (LLMs) has become a major research focus with frontier models providing 1M token context length as standard [5, 38, 26, 10]. For complex tasks, longer contexts enable longer chain-of-thoughts in reasoning models which can improve performance [18]. In RAG applications, longer context lengths complement retrieval mechanisms, allowing models to consume larger numbers of documents at inference time [21].

Mechanistic interpretability seeks to reverse engineer neural networks such as LLMs, often by studying smaller toy models such as GPT-2, or decomposing larger models into simpler units that can be analyzed independently [27]. However, these common techniques such as those introduced by Lindsey et al. [22] and Nanda [27] face scaling challenges when applied to long context scenarios. First, meaningful signals become increasingly diffuse as they spread across extended contexts, making it difficult to isolate and detect interpretable patterns. Second, the attention mechanism's quadratic scaling - $O(T^2)$ in both computational time and memory usage - creates prohibitive resource demands,

[*]Work done during an internship at Spotify.

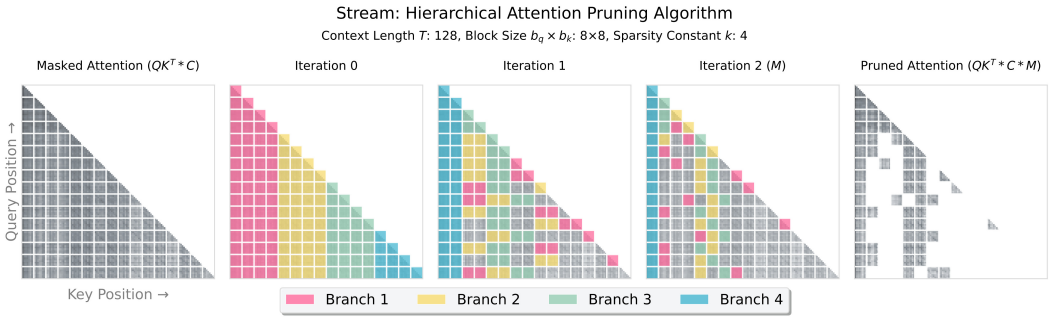

Figure 1: Illustration of the STREAM hierarchical attention pruning algorithm. Starting from the full causally masked attention scores ($QK^T * C$), the pattern is recursively divided into branches and refined over successive iterations. At each step, less relevant regions are discarded, converging on a sparse mask M that preserves only the top-k most relevant key-query interactions. The algorithm produces a final pruned attention score pattern ($QK^T * C * M$), achieving near-linear time complexity $O(TlogT)$ and linear space complexity $O(T)$.

requiring terabytes of memory to cache all attention patterns for contexts with 100,000 tokens or more.

Scaling efforts in mechanistic interpretability have predominantly focused on accommodating larger model parameters, while the impact of extended context lengths remains significantly underexplored. Many recent studies explicitly defer interpretability analysis of contexts exceeding 100 tokens to future work [28, 22, 33], highlighting this critical gap in the field. This paper directly addresses these scalability challenges by introducing an interpretability technique that seamlessly scales to contexts of 100,000 tokens on consumer-grade GPUs, helping democratize long context mechanistic interpretability in realistic settings.

Our key contributions are listed as follows:

- **Scalable long context interpretability.** We introduce SPARSE TRACING, the first framework designed to analyze million-token contexts with its instantiation achieving near-linear time $O(T \log T)$ and linear memory $O(T)$, reducing resource costs by up to four orders of magnitude compared to dense methods.
- **Stream: an efficient and flexible algorithm.** We propose STREAM, a compilable hierarchical pruning algorithm that dynamically estimates sparse attention masks while allowing fine-grained control over interpretability resolution (e.g. sentence-level vs. paragraph-level).
- **Broad validation and impact.** Through chain-of-thought reasoning and RULER benchmarks, we show that STREAM prunes 90-99% of attention links while highlighting critical thought anchors and retrieval paths, enabling long-context mechanistic interpretability on consumer-grade GPUs.

2 Background

Extending the context length of LLMs. The standard vanilla self-attention computation on current hardware accelerators is memory-bound, scaling with $O(T^2)$ where T is context length. For example, consider a standard LLM using `bfloat16` quantization and context length of 100,000 tokens. A single attention pattern would require 20GB of VRAM to realize. Caching all attention patterns across the 34 layers and 8 heads in Gemma 3 4B [32] would require 3.84TB.

FlashAttention [11] is an exact attention algorithm that is computationally equivalent to vanilla attention but achieves $O(T)$ complexity by chunking the $T \times T$ attention pattern into blocks and computing them sequentially. Other approaches achieve sub-quadratic memory complexity by only computing a subset of each attention pattern, applying a sparse mask in three different ways: (1) the popular static sliding window patterns seen in LLMs designed for long context [5, 38, 32], (2) natively trainable sparse attention (NSA) [37] and (3) dynamically generated sparse attention at inference time [20, 39].

A lesser challenge LLMs face as context lengths increase is the quadratically scaled memory footprint of the Key-Value (KV) cache during inference. Grouped-Query Attention (GQA) [1] allows multiple query heads to share a single key and value head, significantly compressing the cache size with minimal impact on performance. As such, this is often not the bottleneck as context lengths increase. Additionally, long context LLMs are often pretrained on shorter contexts and Rotary Positional Embeddings (RoPE) [31] extensions such as YaRN [29], NTK-Aware interpolation [30] and position interpolation [9] are used to extend their context length with no impact on time and space complexity [41].

Hierarchically Pruned Attention (HiP) Attention. Lee et al. [20] introduce Hierarchically Pruned (HiP) Attention to address the quadratic scaling challenges introduced by vanilla attention. The transformer's self-attention layer computes a dense score matrix:

$$S = QK^\top, \qquad P = \mathrm{softmax}(S), \qquad O = PV, \tag{1}$$

with query, key, and value matrices $Q, K, V \in \mathbb{R}^{T\times d}$ for a context of length T. Because S contains all T^2 pairwise interactions, both memory use and runtime scale quadratically, quickly becoming the bottleneck for long contexts. The HiP attention algorithm dynamically generates a sparse attention mask at inference time that retains only the top-k keys per query:

$$M = \text{top_k_mask}(QK^\top), \qquad \tilde{S} = \mathrm{mask}_M(QK^\top), \qquad \tilde{P} = \mathrm{softmax}(\tilde{S}), \qquad O = \tilde{P}V. \tag{2}$$

Here $M \in \{0,1\}^{T\times T}$ is a binary mask with exactly k ones in every row. With only kT non-zeros, the softmax and context projection drop to $O(TlogT)$ time and $O(T)$ memory complexity. Important hyper-parameters are the sparsity level k and the query and key block sizes (b_q, b_k) that balance locality and throughput. The full HiP attention mask estimation algorithm is provided in Appendix B for reference.

3 Method

SPARSE TRACING; a new technique. In this paper we introduce SPARSE TRACING, a technique comprising a set of mechanistic interpretability algorithms that leverage sparse attention to efficiently analyze attention patterns in long context scenarios. Sparse attention methods reduce computational complexity of the attention computation by computing only the most relevant portions of attention patterns. We hypothesize this may be a useful method of identifying parts of the model related to the model output. Additionally, we observe that the same computational bottlenecks that affect attention computation during inference also hinder interpretability analysis. Additionally, many mechanistic interpretability techniques scale linearly with the number of model components being explored, or require multiple forward and/or backwards passes [27, 19, 25]. Our technique explores all components at once, and requires only a single forward pass when the desired sparsity constant k is known, and $\lfloor log_2 k \rfloor$ forward passes otherwise, where the wall clock time of a single forward pass with our chosen SPARSE TRACING algorithm is less than two forward passes without hooks.

STREAM; a new algorithm. We introduce STREAM, our specific SPARSE TRACING algorithm based heavily on HiP attention [20]. STREAM is visualized in Figure 1 and the algorithm is provided in full in Appendix C. HiP attention is a compilable, dynamic sparse attention algorithm that achieves $O(T \log T)$ time complexity and $O(T)$ space complexity and allows the granularity of the attention computation to be tuned. STREAM is heavily based on HiP attention and operates via the same hierarchical search process that efficiently identifies the most relevant blocks of keys for each block of queries.

The algorithm begins by dividing the pattern into blocks of size b_q for queries and b_k for keys, then uses a binary search-inspired approach to progressively narrow down to the top-k most relevant key blocks. The pruning process works in iterations, where each iteration refines the search space by half. Initially, STREAM divides the entire key pattern into k equally-sized branches. The algorithm then identifies which valid branches contain the highest-scoring attention patterns and discards the less relevant regions. In subsequent iterations, STREAM recursively subdivides the remaining promising branches, continuing this hierarchical refinement until it converges on the final set of k key blocks of dimension $b_q \times b_k$ that exhibit the strongest attention relationships with each query block. The behavior of STREAM is governed by the 3 key parameters: the query block size b_q, the

key block size b_k, and the sparsity constant k. Figure 10 illustrates how varying these parameters changes the structure of the resulting sparse attention mask. The granularity of STREAM can be adjusted semantically by setting block sizes to approximate linguistic units (e.g., $b_q = b_k = 32$ for sentence-level attention, $b_q = b_k = 128$ for paragraph-level attention). The sparsity level is controlled by setting the sparsity constant $k \in [1, \left\lfloor \frac{T}{b_k} \right\rfloor]$ which controls the level of pruning. Our SPARSE TRACING method STREAM generates sparse masks for each attention head across all layers. For a given input, we:

1. Compute hierarchical sparse attention masks using STREAM with specified parameters (b_q, b_k, k).
2. Apply these masks to the full attention patterns to identify the most relevant attention connections.
3. Perform binary search over sparsity constant k to find the lowest k that preserves $n_{\text{match}} = 2$ original output tokens, ensuring the sparse pattern captures behaviorally relevant attention.

Our criteria of $n_{\text{match}} = 2$ matching tokens generated in a row is a proxy for perplexity and model performance. We ablate this choice in Appendix A.2. Lee et al. [20] advise against replacing the first few decoder layers l_d of the transformer with their sparse HiP attention as they found that the first few layers have substantially denser attention patterns and pruning these can degrade performance severely. Through ablation studies they recommend $l_d = 3$ which we ablate in Appendix A.1.

4 Case Studies

4.1 Case Study 1: Thought Anchors

Bogdan et al. [6] introduce black-box and white-box approaches to interpreting long context chain-of-thought reasoning traces by aggregating attention patterns between pairs of sentences rather than between pairs of tokens. They coin the term "thought anchors", which are steps in the reasoning trace that have a disproportionately large influence on the remainder of the trace. In this case study, we validate SPARSE TRACING as a highly scalable method - in both time and space complexity - of identifying thought anchors.

Experimental Setup. We follow the experimental setup of Bogdan et al. [6]. STREAM can be applied to any LLM that implements attention, and for this case study we report results on DeepSeek R1-Distill Qwen-1.5B [14, 36] a small reasoning in model in the same family as Bogdan et al. [6]'s prior work. Matching Bogdan et al. [6], we use a temperature of 0.6 and a top-p value of 0.95 to generate responses to problems in the MATH-500 dataset [15]. In a similar manner to Bogdan et al. [6], we use the results of running the model on 500 problems, 10 times each to identify 10 questions that the model correctly solves 25-75% of the time. Venhoff et al. [34] identifies a selection of reasoning behaviors exhibited by DeepSeek-R1-Distill models that can be controlled using steering vectors. Based on this framework, Bogdan et al. [6] prompt an LLM to label sentences as one of 8 distinct categories. These categories are (1) problem setup, (2) plan generation, (3) fact retrieval, (4) active computation, (5) uncertainty management, (6) result consolidation, (7) self-checking and (8) final answer emission. In our experiments, instead of splitting the reasoning trace into sentences, we split the traces into 32 token chunks to match the query block size (b_q) in STREAM. We prompt OpenAI GPT-4o (April-May, 2025) [17] to label each sentence using an almost identical prompt given by Bogdan et al. [6] which we provide in Appendix D.1.

4.1.1 Receiver Heads

Bogdan et al. [6] suggest that "thought anchors" attract higher attention from later parts of the text. In SPARSE TRACING, we adopt a similar idea: salient query blocks should focus on a small, highly relevant set of key blocks. Formally, let

$$\mathcal{Q}_i = Q_{i:i+b_q-1}, \qquad \mathcal{K}_j = K_{j:j+b_k-1} \tag{3}$$

be the ith query and jth key blocks of sizes b_q and b_k. If

$$\alpha(\mathcal{Q}_i, \mathcal{K}_j) = \max_{u \in \mathcal{Q}_i, v \in \mathcal{K}_j} \langle u, v \rangle \tag{4}$$

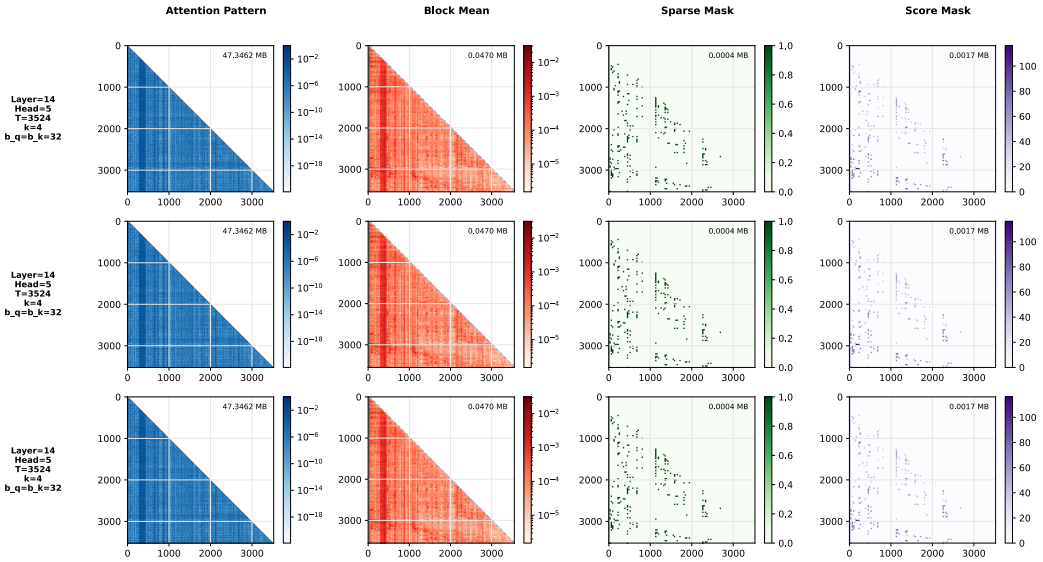

Figure 2: This figure displays attention patterns for three different context lengths (3,000, 6000, and 10000 tokens) using three different approaches: Full Attention Pattern (first column), Mean Averaged [6] with $b = 32$ (second column), and STREAM with $b_y = b_k = 32$ (third and fourth column).

is the maximum token-level dot product, then

$$\Pr\Big[\alpha(\mathcal{Q}_i, \mathcal{K}_j) \text{ ranks in top-}k\Big] \;\propto\; \text{salience}(\mathcal{Q}_i) \tag{5}$$

That is, more salient queries appear more often among top-k pairs. To detect such anchors, we look for attention heads with high kurtosis, meaning a few parts of the context receive much stronger attention than the rest. Following Bogdan et al. [6], we call these receiver heads. They concentrate attention on specific blocks instead of spreading it broadly. Instead of aggregating attention to the sentence level, we compute block-level averages over $b_q \times b_k$ spans, yielding the "block mean" pattern (Figure 2). We select potential receiver heads by measuring the maximum vertical kurtosis of these block means at multiple context lengths. For each candidate head, we generate sparse masks with STREAM, both binary and score-weighted, and visualize them in Figure 2.

Discussion. Across reasoning traces of length 3000, 6000, and 10,000 tokens, SPARSE TRACING consistently prunes 97-99% of attention links while preserving the model's output for two consecutive tokens. This makes the sparse masks 28,000-68,000× more memory efficient than storing dense patterns, with complexity scaling linearly in context length (see Appendix A.3). Crucially, the peaks in the vertical attention distributions reveal potential thought anchors: specific blocks that disproportionately attract attention from downstream queries. These anchors remain visible despite aggressive pruning, showing that SPARSE TRACING highlights the same key structures as the dense patterns but at far lower cost.

4.1.2 Sentence Category Importance

In Figure 3, we compare the sentence category importance based on vertical attention scores, comparing Bogdan et al. [6]'s approach and STREAM.

Discussion. Figure 3 matches the peaks in the vertical scores to identifiable reasoning categories. We observe that problem setup and final answer emission blocks consistently receive the strongest downstream focus, likely due to attention sinks at the beginning and strong diagonal effects near the end of context. In longer traces, receiver heads also peak around plan generation, uncertainty management, and self-checking, while in shorter traces, active computation dominates instead. Overall, we found receiver heads harder to identify as context scales - reporting different heads at different context lengths - and indicating a need for future research to support our results.

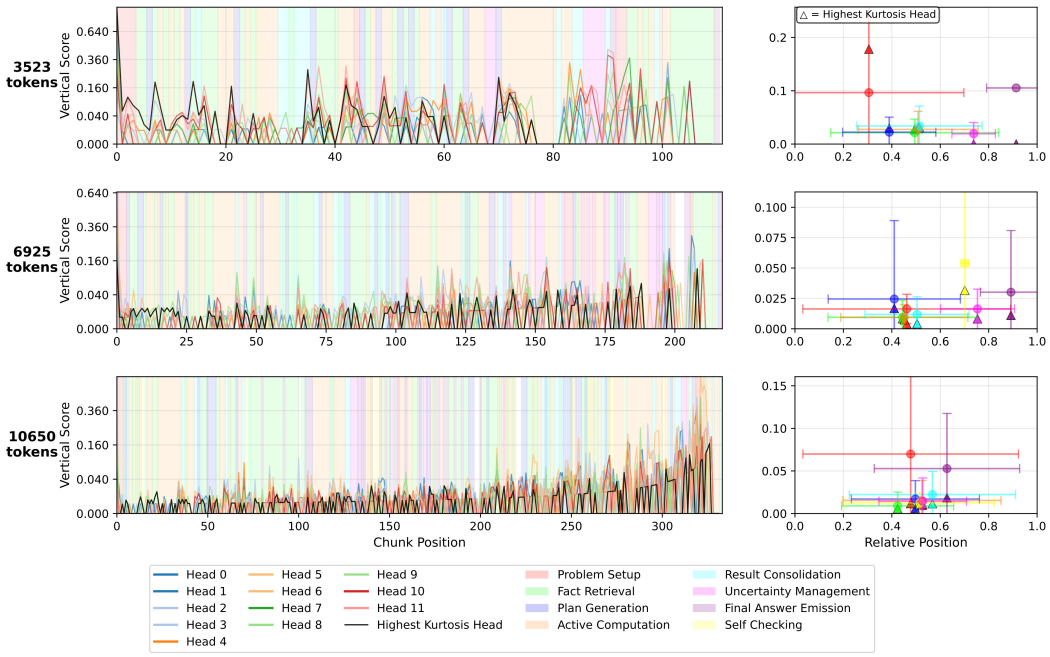

Figure 3: In this figure we plot the vertical attention scores for each 32 token block for three different context lengths (1,000, 10000, and 20,000 tokens).

4.2 Case Study 2: Needle in a Haystack

In this case study, we explore the scalability of STREAM by applying it to the RULER needle in a haystack benchmark [16]. STREAM can be applied to any LLM that implements attention, and we validate the transferability of our method by applying it to a different LLM, namely Gemma 3 1B [32].

Experimental Setup. The experimental data was prepared using the RULER benchmark's [16] synthetic data generation pipeline, which creates structured evaluation samples with controllable context lengths and needle placement positions. For each evaluation instance, we employed a binary search algorithm to determine the optimal sparsity parameter k - the minimum number of key-value blocks required for the sparse attention mechanism to successfully retrieve two consecutive tokens from the needle (see Appendix A.2 for ablations).

4.2.1 Sparse Tracing Needles

In Figure 4 we visualize the needle at 3 different context lengths (1000, 10000 and 20000 tokens). We extract the full attention patterns from a Gemma 3 1B model at the layer and attention head with highest block mean kurtosis.

Discussion. The resulting sparse masks are strikingly structured: despite discarding over 90-96% of attention links, the critical path to the needle remains intact and is clearly visible in the pruned attention maps. This demonstrates STREAM's ability to aggressively reduce computational complexity while still preserving the essential signal required for successful retrieval, even as context length scales by an order of magnitude.

4.2.2 Exploring the effect of Needle Depth

In needle-in-a-haystack tasks, models often show a U-shaped retrieval curve [23]: they are relatively good at retrieving needles near the start of the context (due to over-squashing) and near the end (as those positions are directly learned), but struggle in the middle. By introducing the effective sparsity

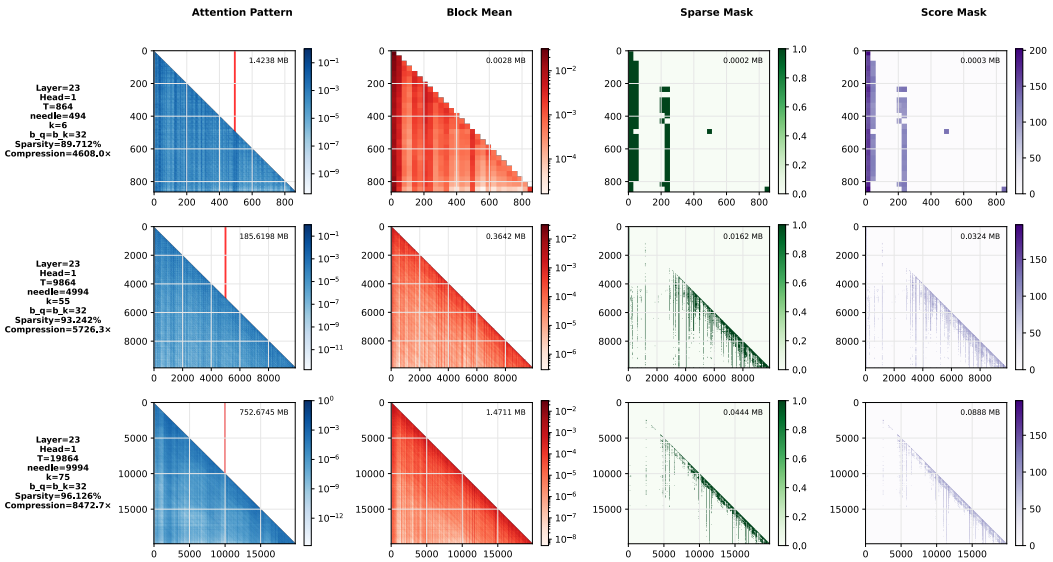

Figure 4: Visualization of sparse attention masks generated by STREAM on the RULER needle-in-a-haystack benchmark [16] at three context lengths (1k, 10k, and 20k tokens). The needle is marked using a red vertical line in the Attention Pattern plots.

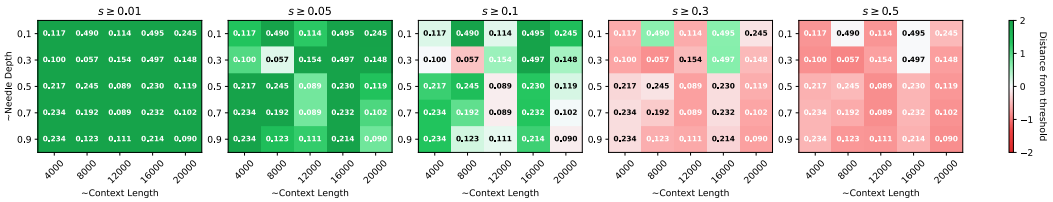

Figure 5: Figure showing needle retrieval success as a function of context length (x-axis), needle depth (y-axis) and effective sparsity s (subplots-axis). The parameter $s \in (0, 1)$ controls the effective pruning of the attention pattern via k with larger s corresponding to higher pruning.

s such that:

$$k = \left\lfloor 1 + \left(\frac{T}{b_q} - 1 \right) \cdot s \right\rfloor \quad \text{such that} \quad 1 \leq k \leq \left\lfloor \frac{T}{b_q} \right\rfloor \quad \text{and} \quad 0 < s < 1 \tag{6}$$

The larger the s, the higher the pruning of the attention pattern. Figure 5 visualizes successful needle retrieval as s, context length and needle depth vary. We see that the retrieval of needles earlier in the context is more robust as context length scales.

Discussion. Instead of the characteristic U-shape, we find that STREAM's attention pruning weakens needle retrieval towards the end of longer contexts with effective sparsity $s \geq 0.1$, leading to failures above 8000 tokens. The lower triangular nature of causal attention patterns results in a larger number of valid candidate key blocks per query towards the bottom of the pattern. This leads to a higher level of pruning at constant k, which would be resolved by implementing variable k in future research.

4.2.3 Exploring Information Flow

Existing methods for tracing information flow in transformers [13, 27, 19, 25] scale quadratically with context length. To our knowledge, STREAM is the first such technique with linear scaling, making it feasible for tasks like indirect object identification and needle-in-a-haystack. For the latter, we report sparse patterns at $T = 1000$, needle depth $n_d \approx 0.5$, $b_q = b_k = 32$, with $k_{\text{success}} = 6$ (successful retrieval) and $k_{\text{fail}} = 3$ (unsuccessful retrieval). Subtracting the k_{fail} masks from k_{success} across layers and heads isolates the connections that enable retrieval. Figure 6 visualizes these paths,

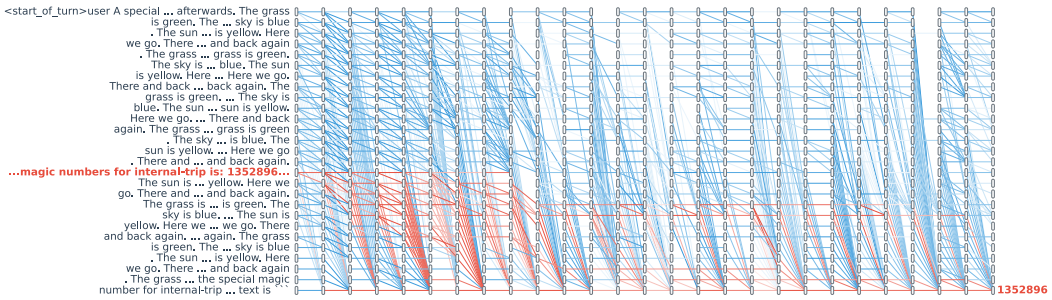

Figure 6: Attention flow paths distinguishing successful and unsuccessful retrieval in the needle-in-a-haystack task ($T = 1000$, $n_d \approx 0.5$). Each node represents a hidden state block of size $b_q = b_k = 32$. Red edges trace paths carrying needle information to the output, while blue edges denote other attention flows. Paths are weighted by block scores of k_{success}, and obtained by subtracting the sparse masks of $k_{\text{fail}} = 3$ from $k_{\text{success}} = 6$.

weighted by block scores: red edges trace needle-to-output flows, while blue edges denote other attention routes. We restrict analysis to attention paths, leaving residual and MLP components for future SPARSE TRACING extensions. Full patterns for k_{success} and k_{fail} are shown in Appendix E.1.

Discussion. We see that the needle information appears to travel along two paths to the output, through hidden state blocks 18 and 19, and through the final few blocks. We also see attention sink behavior - high attention to earlier tokens in the context - when needle information passes to the output block, which may be preventing over-mixing, a form of rank collapse [4]. This would aid in preserving the information from the needle through the layers of the network. This experiment consolidates the use of STREAM as an empirical tool to explore theoretical claims.

5 Related Work

Chain-of-Thought White-box Monitoring. Reasoning models produce a long chain of thought before responding to the user, with the aim of increasing their performance on complex tasks [18]. A growing branch of Mechanistic Interpretability research seeks to understand model internals during chain-of-thought processes [6, 8, 7, 12, 34]. Brinkmann et al. [7] demonstrate that transformers can implement interpretable backward chaining algorithms for pathfinding, while Dutta et al. [12] extend this understanding to production-scale models, revealing multiple parallel pathways for answer generation. Venhoff et al. [34] show these reasoning behaviors can be controlled through steering vectors. Cabannes et al. [8] find identifiable computational structures that enable iterative algorithms, and Bogdan et al. [6] take black-box and white-box approaches to identify "thought anchors" - the most critical parts of the reasoning trace. In our work, we use our SPARSE TRACING methodology to scalably identify "thought anchors" in longer contexts.

Why LLMs struggle with Long Context. Other work examines how LLMs process extended contexts and the mechanisms underlying their successes and failures [35, 23, 3, 2, 24]. Wu et al. [35] identify "retrieval heads" - a sparse set of attention heads (3-6%) specifically responsible for copying information from arbitrary locations in long contexts, providing a mechanistic explanation for factual retrieval capabilities. This work builds on Liu et al. [23]'s empirical finding that models perform poorly when relevant information appears in the middle of long contexts. Barbero et al. [2] provide theoretical grounding for these limitations, proving that decoder-only transformers suffer from "information over-squashing" where distinct input contexts become indistinguishable in final token representations. Their companion work [3] reveals how rotary positional embeddings can mitigate some issues by using different frequency components for positional versus semantic attention. Our work introduces STREAM, which builds highly scalable computation graphs for information propagation research into Geometric Deep Learning metrics such as over-smoothing, over-squashing and under-reaching.

Scaling Mechanistic Interpretability. Tooling has been built for conducting Mechanistic Interpretability research as model size, complexity and context lengths scale [27, 19, 13, 40]. Nanda [27] introduces attribution patching, enabling activation patching experiments on large models through gradient-based approximations. Kramár et al. [19] address failure modes in this approach, proposing AtP*. Ferrando and Voita [13] develop automated methods for discovering information flow routes without requiring human-designed counterfactuals, while Zheng et al. [40] provide systematic frameworks for understanding attention head functions. Paulo et al. [28] address the scalability challenge of interpreting sparse autoencoder (SAE) features by developing an automated pipeline that can generate and evaluate natural language explanations for millions of features. Templeton et al. [33] successfully scale SAEs to Claude 3 Sonnet extracting millions of interpretable features. We introduce the first Mechanistic Interpretability tool designed to run on consumer hardware that scales to long context.

6 Limitations

(1) STREAM focuses on the attention mechanism and makes no causal claims. Residual connections and MLP layers also play crucial roles in computation, and STREAM does not currently account for these. (2) The method's success is defined by a proxy metric, specifically maintaining two consecutive correct tokens in the output. This is a practical heuristic but may not fully capture the nuance of model performance or the complete causal path. (3) While STREAM achieves impressive computational complexity which holds as contexts scale, our current implementation has not been optimized for specific hardware and therefore runs slower than desired. (4) The pruning algorithm is more aggressive towards the bottom of the attention pattern, which could be resolved by implementing a variable sparsity constant k. (5) Future work introducing a novel discovery made by STREAM would truly validate our method's applicability.

7 Conclusion

Our work introduces SPARSE TRACING and its instantiation, STREAM, as scalable tools for mechanistic interpretability in long context LLMs. These methods address a key limitation of existing interpretability techniques: the quadratic complexity of attention, which makes direct analysis of million-token contexts practically infeasible. By leveraging hierarchical pruning, STREAM achieves near-linear $O(T \log T)$ time and linear $O(T)$ space complexity while preserving salient attention pathways, enabling experiments previously restricted to short contexts. We apply STREAM to both reasoning (DeepSeek R1-Distill Qwen-1.5B) and non-reasoning models (Gemma 3 1B) and through 5 experiments and 3 ablations, we validate the widespread applicability of our method. In chain-of-thought reasoning at 10,000 token context, SPARSE TRACING prunes 99% of the attention pattern and identifies influential "thought anchors" requiring 60,000× less memory than caching the full patterns. In needle-in-a-haystack tasks, STREAM highlights the sparse structures required for needle retrieval, pruning the pattern by 96% at 20,000 token context.

Future Work. (1) Future work should incorporate MLP layers and residual stream analysis into the SPARSE TRACING framework providing a more complete picture of information flow. (2) Establishing theoretical guarantees about what types of circuits and information pathways are preserved under different sparsity levels would strengthen confidence in sparse interpretability results. (3) STREAM opens new possibilities for studying long context phenomena like information over-squashing, over-smoothing, and under-reaching. (4) Further work could also explore the effect of aligning block sizes with semantic units such as sentences or paragraphs and extend STREAM to further case studies.

Acknowledgments and Disclosure of Funding

J Rosser is supported by the EPSRC Centre for Doctoral Training in Autonomous and Intelligent Machines and Systems EP/Y035070/1. This work was carried out during an internship at Spotify, who also provided the computational resources. We are especially grateful to Federico Barbero, whose insight and early guidance were instrumental in shaping the direction of this research, and who provided invaluable technical input throughout the project. We also thank Mounia Lalmas-Roelleke, Paul Bennett, Ali Vardasbi, Enrico Palumbo, and Jiazheng Li for generously providing thoughtful feedback and discussions.

References

[1] Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

[2] Federico Barbero, Andrea Banino, Steven Kapturowski, Dharshan Kumaran, João Madeira Araújo, Oleksandr Vitvitskyi, Razvan Pascanu, and Petar Veličković. Transformers need glasses! information over-squashing in language tasks. *Advances in Neural Information Processing Systems*, 37:98111–98142, 2024.

[3] Federico Barbero, Alex Vitvitskyi, Christos Perivolaropoulos, Razvan Pascanu, and Petar Veličković. Round and round we go! what makes rotary positional encodings useful? *arXiv preprint arXiv:2410.06205*, 2024.

[4] Federico Barbero, Alvaro Arroyo, Xiangming Gu, Christos Perivolaropoulos, Michael Bronstein, Petar Veličković, and Razvan Pascanu. Why do llms attend to the first token? *arXiv preprint arXiv:2504.02732*, 2025.

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

[6] Paul C Bogdan, Uzay Macar, Neel Nanda, and Arthur Conmy. Thought anchors: Which llm reasoning steps matter? *arXiv preprint arXiv:2506.19143*, 2025.

[7] Jannik Brinkmann, Abhay Sheshadri, Victor Levoso, Paul Swoboda, and Christian Bartelt. A mechanistic analysis of a transformer trained on a symbolic multi-step reasoning task. *arXiv preprint arXiv:2402.11917*, 2024.

[8] Vivien Cabannes, Charles Arnal, Wassim Bouaziz, Xingyu Yang, Francois Charton, and Julia Kempe. Iteration head: A mechanistic study of chain-of-thought. *Advances in Neural Information Processing Systems*, 37:109101–109122, 2024.

[9] Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*, 2023.

[10] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

[11] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

[12] Subhabrata Dutta, Joykirat Singh, Soumen Chakrabarti, and Tanmoy Chakraborty. How to think step-by-step: A mechanistic understanding of chain-of-thought reasoning. *arXiv preprint arXiv:2402.18312*, 2024.

[13] Javier Ferrando and Elena Voita. Information flow routes: Automatically interpreting language models at scale. *arXiv preprint arXiv:2403.00824*, 2024.

[14] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[15] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

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

[17] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[18] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

[19] János Kramár, Tom Lieberum, Rohin Shah, and Neel Nanda. Atp*: An efficient and scalable method for localizing llm behaviour to components. *arXiv preprint arXiv:2403.00745*, 2024.

[20] Heejun Lee, Geon Park, Youngwan Lee, Jina Kim, Wonyoung Jeong, Myeongjae Jeon, and Sung Ju Hwang. Hip attention: Sparse sub-quadratic attention with hierarchical attention pruning. *arXiv e-prints*, pages arXiv–2406, 2024.

[21] Zhuowan Li, Cheng Li, Mingyang Zhang, Qiaozhu Mei, and Michael Bendersky. Retrieval augmented generation or long-context llms? a comprehensive study and hybrid approach. *arXiv preprint arXiv:2407.16833*, 2024.

[22] Jack Lindsey, Wes Gurnee, Emmanuel Ameisen, Brian Chen, Adam Pearce, Nicholas L. Turner, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. On the biology of a large language model. *Transformer Circuits Thread*, 2025. URL https://transformer-circuits.pub/2025/attribution-graphs/biology.html.

[23] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023.

[24] Yijun Liu, Jinzheng Yu, Yang Xu, Zhongyang Li, and Qingfu Zhu. A survey on transformer context extension: Approaches and evaluation. *arXiv preprint arXiv:2503.13299*, 2025.

[25] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *Advances in neural information processing systems*, 35:17359–17372, 2022.

[26] AI Meta. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation. *https://ai. meta. com/blog/llama-4-multimodal-intelligence/, checked on*, 4(7):2025, 2025.

[27] Neel Nanda. Attribution patching: Activation patching at industrial scale. *URL: https://www. neelnanda. io/mechanistic-interpretability/attribution-patching*, 2023.

[28] Gonçalo Paulo, Alex Mallen, Caden Juang, and Nora Belrose. Automatically interpreting millions of features in large language models. *arXiv preprint arXiv:2410.13928*, 2024.

[29] Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*, 2023.

[30] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[31] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[32] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

[33] Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L Turner, Callum McDougall, Monte MacDiarmid, C. Daniel Freeman, Theodore R. Sumers, Edward Rees, Joshua Batson, Adam Jermyn, Shan Carter, Chris Olah, and Tom Henighan.

Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *Transformer Circuits Thread*, 2024. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html.

[34] Constantin Venhoff, Iván Arcuschin, Philip Torr, Arthur Conmy, and Neel Nanda. Understanding reasoning in thinking language models via steering vectors. *arXiv preprint arXiv:2506.18167*, 2025.

[35] Wenhao Wu, Yizhong Wang, Guangxuan Xiao, Hao Peng, and Yao Fu. Retrieval head mechanistically explains long-context factuality. *arXiv preprint arXiv:2404.15574*, 2024.

[36] An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

[37] Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, YX Wei, Lean Wang, Zhiping Xiao, et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *arXiv preprint arXiv:2502.11089*, 2025.

[38] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33: 17283–17297, 2020.

[39] Jintao Zhang, Chendong Xiang, Haofeng Huang, Jia Wei, Haocheng Xi, Jun Zhu, and Jianfei Chen. Spargeattn: Accurate sparse attention accelerating any model inference. *arXiv preprint arXiv:2502.18137*, 2025.

[40] Zifan Zheng, Yezhaohui Wang, Yuxin Huang, Shichao Song, Mingchuan Yang, Bo Tang, Feiyu Xiong, and Zhiyu Li. Attention heads of large language models: A survey. *arXiv preprint arXiv:2409.03752*, 2024.

[41] Meizhi Zhong, Chen Zhang, Yikun Lei, Xikai Liu, Yan Gao, Yao Hu, Kehai Chen, and Min Zhang. Understanding the rope extensions of long-context llms: An attention perspective. *arXiv preprint arXiv:2406.13282*, 2024.

A Ablations

A.1 Ablation Study of Dense Layers

To validate $l_d = 3$, we perform a similar ablation study as in Appendix D.5 in Lee et al. [20]'s study and we report results in Figure 7.

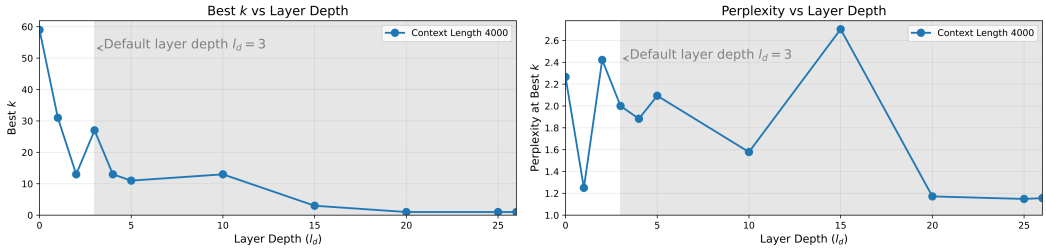

Figure 7: Ablation of dense layers. Left: best sparsity constant k as a function of layer depth. Right: perplexity at the corresponding best k. Results support the choice of leaving the first $l_d = 3$ layers dense, where attention patterns are less sparse and pruning leads to degraded performance.

A.2 Ablation Study of Number of Consecutive Tokens Matching as a Proxy for Perplexity

To maximize pruning efficiency, we aim to select the smallest sparsity constant k that still preserves model performance. Directly measuring degradation via average perplexity is impractical, since it is unclear at what threshold perplexity becomes unacceptable. Instead, we adopt a behavioral proxy: running the model with sparse masks for varying k and checking when it can no longer reproduce the same output. We define performance preservation as generating at least $n_{\text{match}} = 2$ consecutive matching tokens, and report ablations for this criterion in Figure 8. Notably, in the middle and right plots, we observe that once k or average perplexity is large enough to achieve $n_{\text{match}} = 2$, at any k higher the model typically recovers the entire output sequence, suggesting that this threshold is a reliable indicator of stable performance.

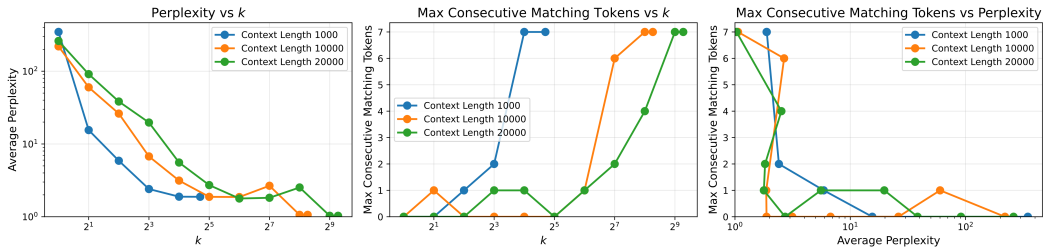

Figure 8: Ablation of sparsity constant k search. Left: average perplexity decreases with increasing k. Middle: number of consecutive matching tokens improves with larger k, with $n_{\text{match}} = 2$ serving as our performance threshold. Once this threshold is met, the model typically produces the full correct output. Right: trade-off between perplexity and matching tokens highlights the balance between pruning aggressiveness and output fidelity.

A.3 Ablation Study of Time and Space Complexity

In Figure 9 we report the GPU Memory Usage and Wall Clock Time as they vary with sparsity level k, context length and block size (b_q, b_k).

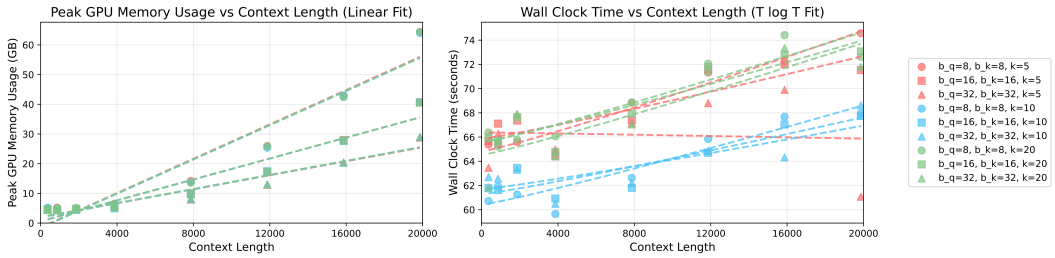

Figure 9: GPU memory usage (linear fits, mean R^2 = 0.9308) and wall clock time (T log T fits, mean R^2 = 0.7282) versus context length for varying block sizes (b_q, b_k) and sparsity levels k.

B HiP Attention Algorithm

In Algorithm 1 we provide the full Hierarchical Sparse Attention Mask Estimation algorithm as presented by Lee et al. [20].

Algorithm 1: Hierarchical Sparse Attention Mask Estimation

Input: Queries $\boldsymbol{Q} \in \mathbb{R}^{T \times d}$, Keys $\boldsymbol{K} \in \mathbb{R}^{T \times d}$, Sparsity constant k, Query block size b_q, Key block size b_k, Top-r approximation constant r

Output: Estimated attention mask $\widehat{\boldsymbol{M}} \in \{0,1\}^{T \times T}$ which is represented by an array of indices $\mathcal{I} \in [1:T]^{T/b_q \times k/b_k}$

$\mathbf{Q}, \mathbf{K} = \text{reshape}_{T/b_q \times b_q \times d}[\boldsymbol{Q}], \text{reshape}_{T/b_k \times b_k \times d}[\boldsymbol{K}]$;

$n_{it} = \lceil \log(T/b_k) \rceil$;

`// Number of iterations`

for *each query block index* $q = 1 \,..\, T/b_q$ **do**

  $\left(f_{qj}^{(1)}, l_{qj}^{(1)}\right) = \left(\lfloor j \cdot \frac{T}{k} \rfloor, \lfloor (j+1) \cdot \frac{T}{k} \rfloor - 1\right)$ for $j = 1 \,..\, k$;

  `// Set` k `nodes' initial start and end indices`

  for *each iteration* $i = 1 \,..\, n_{it}$ **do**

    for *each node index* $j = 1 \,..\, k$ **do**

      $m_{qj}^{(i)} = \lfloor (f_{qj}^{(i)} + l_{qj}^{(i)})/2 \rfloor$;

      $\left(\mathcal{B}_{q,2j-1}^{(i)}, \mathcal{B}_{q,2j}^{(i)}\right) = \left((f_{qj}^{(i)} : m_{qj}^{(i)} - 1), (m_{qj}^{(i)} : l_{qj}^{(i)})\right)$;

    for *each branch index* $h = 1 \,..\, 2k$ **do**

      Pick a first index $r_{qh}^{(i)}$ from the range $\mathcal{B}_{qh}^{(i)}$;

      Compute score $s_{qh}^{(i)} = \max_{m,n}\left(\mathbf{Q}_{q,m,:}^{\top} \mathbf{K}_{r_{qh}^{(i)},n,:}\right)$;

    Pick top-k indices $\{t_1, \ldots, t_k\}$ of the sequence $s_{q,1}^{(i)}, \ldots, s_{q,2k}^{(i)}$;

    Update nodes $(f_{qj}^{(i+1)} : l_{qj}^{(i+1)}) := \mathcal{B}_{t_j}^{(i)}$ for $j = 1 \,..\, k$;

  Set mask indices $\mathcal{I}_{qj} = f_{qj}^{(n_{it})}$ for $j = 1 \,..\, k$;

C STREAM Algorithm

We provide the full STREAM algorithm in Algorithm 2. In Figure 10 we visualize the effect of b_q, b_k and k on the final sparse attention mask.

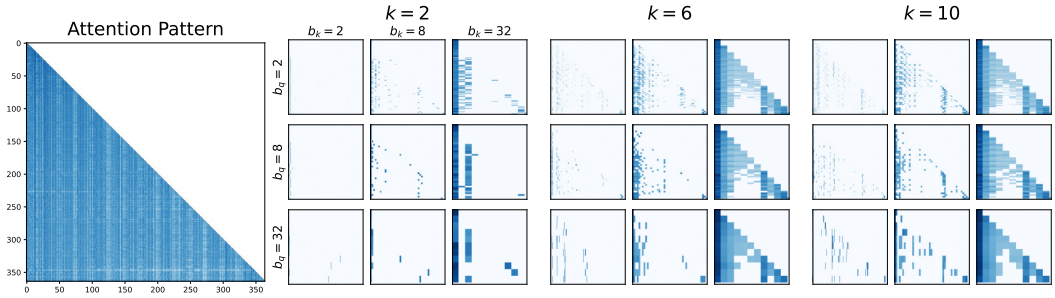

Figure 10: Effect of block and sparsity parameters on sparse attention masks. Increasing b_q and b_k coarsens the mask by aggregating larger query and key spans, while higher values of k preserve more key blocks, yielding denser masks.

Algorithm 2: STREAM

Input: Queries $\boldsymbol{Q} \in \mathbb{R}^{T \times d}$, Keys $\boldsymbol{K} \in \mathbb{R}^{T \times d}$, Attention mask $\boldsymbol{C} \in \{0,1\}^{T \times T}$, Sparsity k, Query block b_q, Key block b_k, Top-r constant r

Output: $\widehat{\boldsymbol{M}} \in \{0,1\}^{T \times T}$ via indices $\mathcal{I} \in [1:T]^{T/b_q \times k/b_k}$

$\ell \leftarrow \mathrm{lcm}(b_q, b_k)$;
$T_p \leftarrow \lceil T/\ell \rceil \cdot \ell$;
$\mathrm{extra} \leftarrow T_p - T$;
$T \leftarrow T_p$;
$Q \leftarrow \mathrm{pad}(Q, ((0, \mathrm{extra}), (0,0)))$, $\quad K \leftarrow \mathrm{pad}(K, ((0, \mathrm{extra}), (0,0)))$;
$C \leftarrow \mathrm{pad}(C, ((0, \mathrm{extra}), (0, \mathrm{extra})))$
$\mathbf{Q}, \mathbf{K} = \mathrm{reshape}_{T/b_q \times b_q \times d}[\boldsymbol{Q}],\ \mathrm{reshape}_{T/b_k \times b_k \times d}[\boldsymbol{K}]$;
$\mathbf{C} = \mathrm{reshape}_{T/b_q \times b_q \times T/b_k \times b_k}[\boldsymbol{C}]$;
$\boldsymbol{C}^{\mathrm{blk}} \in \{0,1\}^{T/b_q \times T/b_k}$, $\boldsymbol{C}^{\mathrm{blk}}[q,r] = \max_{m \le b_q, n \le b_k} \mathbf{C}[q,m,r,n]$;
$n_{it} = \lceil \log(T/b_k) \rceil$;
for $q = 1 \,..\, T/b_q$ **do**
  $\left(f_{qj}^{(1)}, l_{qj}^{(1)}\right) = \left(\lfloor j \frac{T}{k} \rfloor,\ \lfloor (j+1)\frac{T}{k} \rfloor - 1\right)$ for $j = 1 \,..\, k$;
  for $i = 1 \,..\, n_{it}$ **do**
    for $j = 1 \,..\, k$ **do**
      $m_{qj}^{(i)} = \lfloor (f_{qj}^{(i)} + l_{qj}^{(i)})/2 \rfloor$;
      $\left(\mathcal{B}_{q,2j-1}^{(i)}, \mathcal{B}_{q,2j}^{(i)}\right) = \left((f_{qj}^{(i)} : m_{qj}^{(i)} - 1), (m_{qj}^{(i)} : l_{qj}^{(i)})\right)$;
    for $h = 1 \,..\, 2k$ **do**
      $\omega_{qh}^{(i)} = \max\limits_{r \in \mathcal{B}_{qh}^{(i)}} \boldsymbol{C}^{\mathrm{blk}}[q,r]$;
      if $\omega_{qh}^{(i)} = 0$ **then**
        set $s_{qh}^{(i)} = -\infty$ and **continue**
      Pick first $r_{qh}^{(i)} \in \mathcal{B}_{qh}^{(i)}$ with $\boldsymbol{C}^{\mathrm{blk}}[q, r_{qh}^{(i)}] = 1$;
      Compute $s_{qh}^{(i)} = \max\limits_{\substack{m \le b_q,\, n \le b_k \\ \mathbf{C}[q,m,r_{qh}^{(i)},n]=1}} \mathbf{Q}_{q,m,:}^{\top} \mathbf{K}_{r_{qh}^{(i)},n,:}$;
    Pick top-k indices $\{t_1, \ldots, t_k\}$ of $s_{q,1}^{(i)}, \ldots, s_{q,2k}^{(i)}$;
    Update $(f_{qj}^{(i+1)} : l_{qj}^{(i+1)}) := \mathcal{B}_{t_j}^{(i)}$ for $j = 1 \,..\, k$;
  Set $\mathcal{I}_{qj} =$ first $r \in (f_{qj}^{(n_{it})} : l_{qj}^{(n_{it})})$ with $\boldsymbol{C}^{\mathrm{blk}}[q,r] = 1$; if none, leave $\mathcal{I}_{qj}$ empty;

D Thought Anchors: Additional Results

D.1 Sentence Labelling Prompt

Here we repeat the prompt provided by Bogdan et al. [6] to label sentences, with minor modifications.

```
You are an expert in interpreting how language models solve math problems
using multi-step reasoning.  Your task is to analyze a Chain-of-Thought
(CoT) reasoning trace, broken into discrete text chunks, and label each
chunk with the single best **function_tag** that describes what the chunk
is doing.

--

### Function Tags (you may only assign one per chunk):

1.  `problem_setup`:  Parsing or rephrasing the problem (initial reading or
comprehension).

2.  `plan_generation`:  Stating or deciding on a plan of action (often
meta-reasoning).

3.  `fact_retrieval`:  Recalling facts, formulas, problem details (without
immediate computation).

4.  `active_computation`:  Performing algebra, calculations, manipulations
toward the answer.

5.  `result_consolidation`:  Aggregating intermediate results, summarizing,
or preparing final answer.

6.  `uncertainty_management`:  Expressing confusion, re-evaluating,
proposing alternative plans (includes backtracking).

7.  `final_answer_emission`:  Explicit statement of the final boxed answer
or earlier chunks that contain the final answer.

8.  `self_checking`:  Verifying previous steps, Pythagorean checking,
re-confirmations.

9.  `unknown`:  Use only if the chunk does not fit any of the above tags or
is purely stylistic or semantic.

--

### Output Format:

Return a single dictionary with one entry per chunk, where each entry has:
- the chunk index (as the key, converted to a string), - a string with the
tag

Here's the expected format:

```language=json
{
"0":  "plan_generation"
"1":  "fact_retrieval"
"2":  "active_computation"
"3":  "self_checking"
"4":  "final_answer_emission"
...
}
```
Here is the math problem:

[PROBLEM]
{problem_text}
```

```
Here is the full Chain of Thought, broken into chunks:

[CHUNKS]
{full_chunked_text}

Now label each chunk with function tags only.
```

D.2 Full Attention Comparison

We extend Figure 2 and include the vertical attention scores for block mean, sparse mask and score mask for the receiver head.

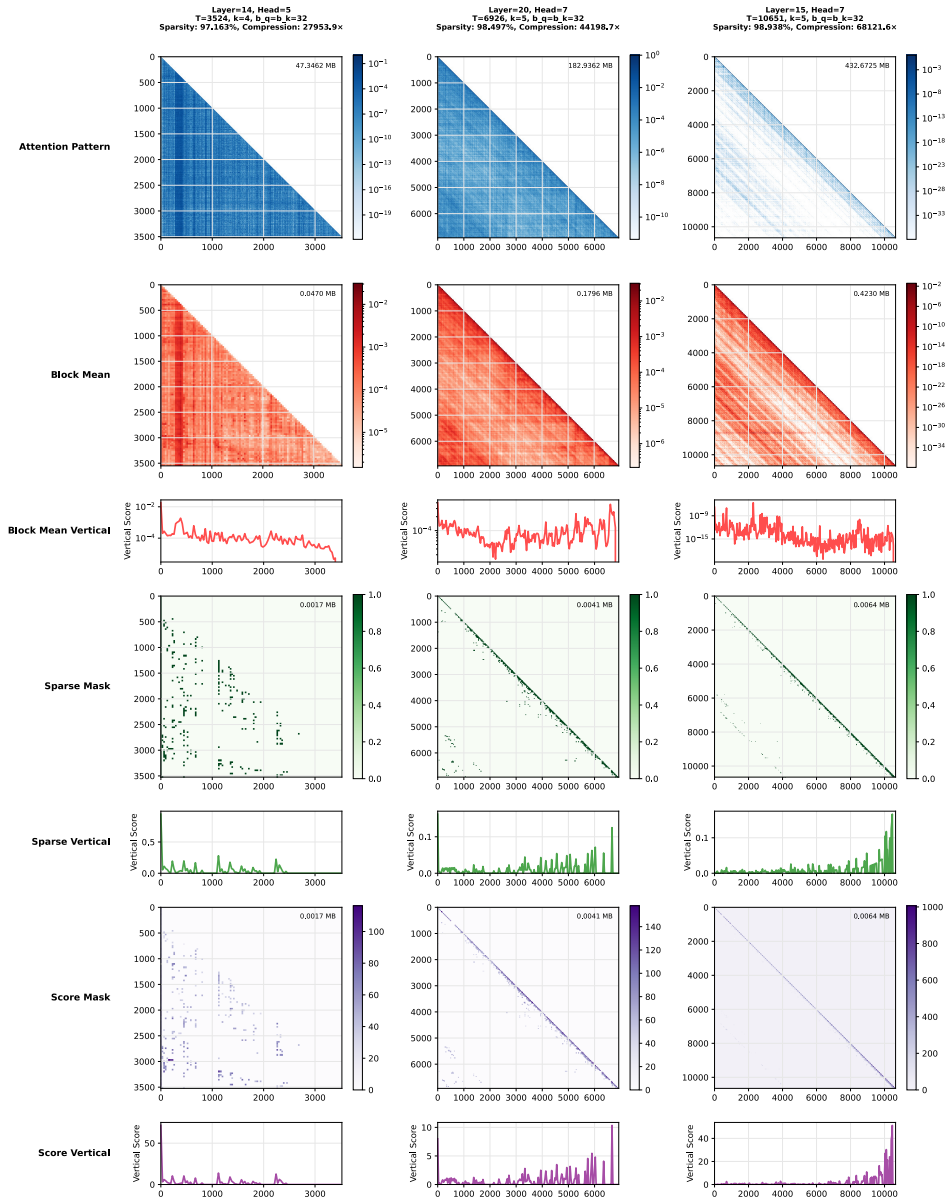

Figure 11: Extending Figure 2 to include the vertical attentions score plots.

D.3 Sentence Category Distribution

We report a similar sentence category distribution to Bogdan et al. [6] as shown in Figure 12.

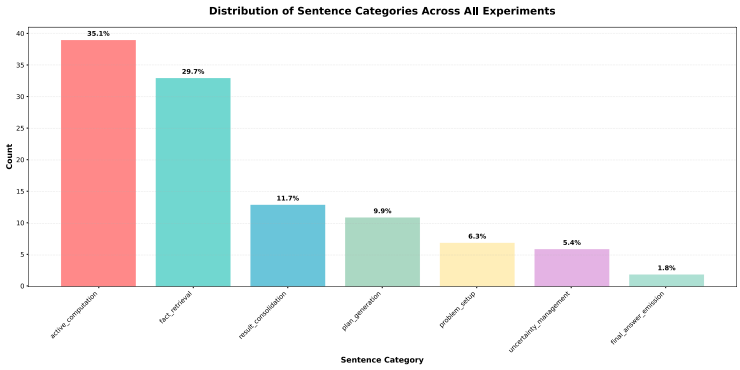

Figure 12: Sentence category vs counts across all experiments.

E Needle-In-A-Haystack: Additional Results

E.1 Information Flow Full Plots

We report the information flow for a successful needle retrieval in Figure 13 and unsuccessful retrieval in Figure 14.

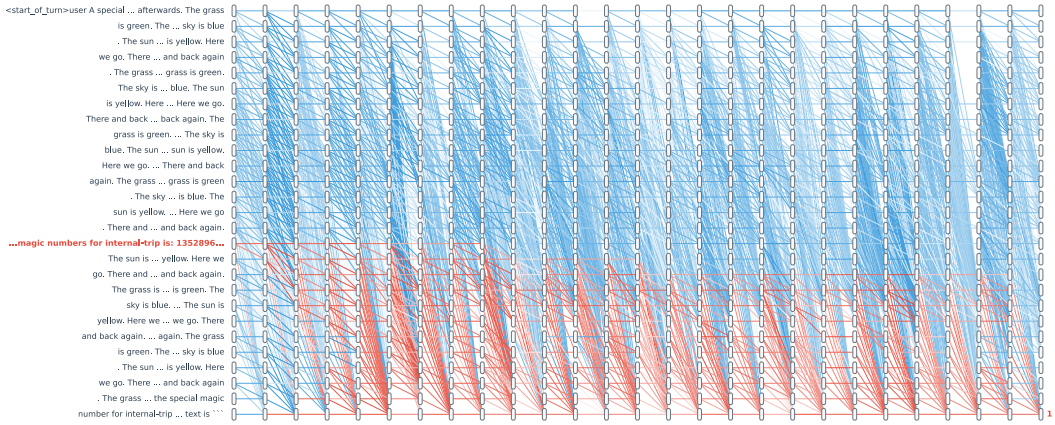

Figure 13: All information paths for a successful needle retrieval ($k = 6$). We plot paths from the needle block to the output in red and the rest in blue.

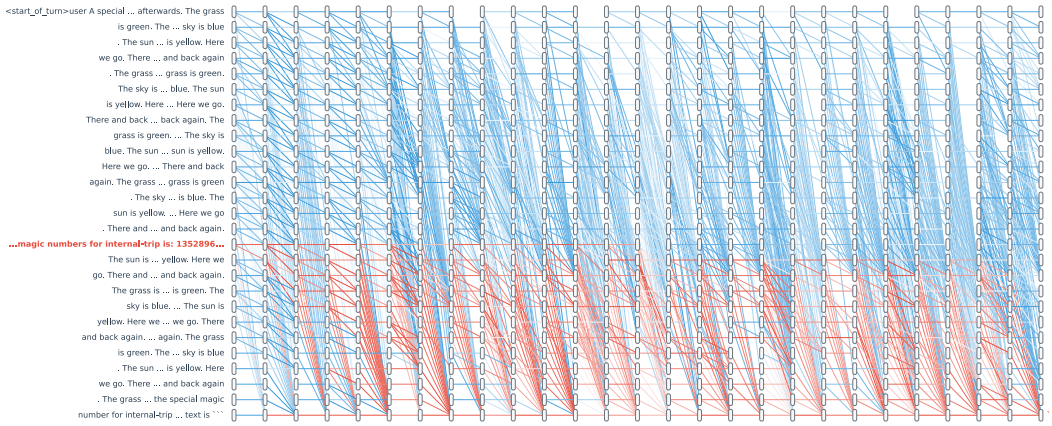

Figure 14: All information paths for a unsuccessful needle retrieval ($k = 3$). We plot paths from the needle block to the output in red and the rest in blue.