Scaling up Mechanistic Interpretability

Hugo Gaible Thomas Winninger

30 May 2026

1 Introduction

LLMs are models that handle a high number of tokens (over a million, even for modest models). Many techniques for studying interpretability have emerged, but they are often limited to miniature models or focused on very specific parts of the LLMs. In general, there is a scaling problem with LLMs, especially when the context length increases. Indeed, the computational and memory complexity of standard attention is $O(T^2)$, so interpretability methods which require analysing these attention patterns can quickly show exponential complexities, which quickly becomes unmanageable when analysing reasoning models with long traces.

The article from (Rosser et al., 2025) proposes to solve this with techniques distributed across multiple GPUs. The algorithm, named Stream, achieves a complexity of $O(T \log T)$ in time and $O(T)$ in memory. With this new algorithm, the calculation of the attention mask is sparse and efficient, with seed-based control for reproducibility. This allows for the analysis of the model while preserving a large amount of essential information.

We implemented their algorithm in PyTorch, using torch's compilation capabilities, replicated some of their results, and performed additional experiments, highlighting the limits of their work.

2 Background

Basic optimizations for attention mechanisms are essential for handling long contexts. We already use FlashAttention, which calculates attention in blocks to optimize memory access, shifting from a heavy management of $O(T^2)$ to a linear management of $O(T)$ for RAM. To save even more resources, there are approaches based on masks (Sparsity): one can use static methods (sliding window), trained parameters to define what to mask, or even dynamic methods that adapt in real-time.

A major challenge of LLMs is the quadratic explosion of memory occupied by the KV Cache (Key-Value) during inference. The article highlights that it is no longer necessarily the main bottleneck thanks to Grouped-Query Attention (GQA). This technique allows multiple query heads to share a single key and value head, thereby compressing the cache size with minimal impact on performance. Moreover, for models trained on short contexts, extensions of Rotary Positional Embeddings (RoPE) such as YaRN, NTK-Aware interpolation, or position interpolation are used. These methods allow extending the length of the context managed by the model without increasing time or space complexity.

The article relies on Hierarchically Pruned Attention (HiP) (Lee et al., 2025) to circumvent the quadratic complexity of classical attention ($S = QK^\top$). The principle is to dynamically generate a binary attention mask $M \in \{0,1\}^{\{T \times T\}}$ during inference. This mask filters the calculations to keep only the top-k best keys for each query. Result: With only k non-zero terms per line, the attention complexity collapses to $O(T \log T)$ in time and $O(T)$ in memory. The balance between speed and accuracy is managed by the sparsity k and the block sizes (b_q, b_k).

3 STREAM Method

3.1 Sparse Tracking

The article introduces Sparse Tracing methods, which encompass techniques that leverage sparse attention not as an optimization, but as an analysis tool. Its advantage is the ability to analyze all the components of the model simultaneously in a single pass, making it much faster than other interpretability methods. Moreover, Sparse Tracing allows mapping the flow of information by identifying the preferred communication channels between the different attention heads and the model layers.

The Sparse Tracing approach is in line with recent work on monosemanticity (Templeton and others, 2023). The major challenge of LLMs is "overlap," where a single neuron simultaneously activates multiple different concepts.

By using attention or sparse autoencoders, we force the model to isolate unitary features. This allows a decomposition of the Residual stream into human-understandable concepts. Furthermore, we can track the activation of a specific concept through the layers, and finally, it can eliminate the "noise" of polysemantic activations that pollute the analysis of the information flow.

3.2 STREAM algorithm

The article introduces its algorithm STREAM, which is a specific sparse tracing algorithm designed to analyze attention over very long contexts. STREAM is actually greatly inspired by HiP, as STREAM is a dynamic algorithm based on HiP's attention, which reduces the complexity to $O(T)$ for memory and $O(T \log T)$ for time, while allowing the computational granularity to be adjusted.

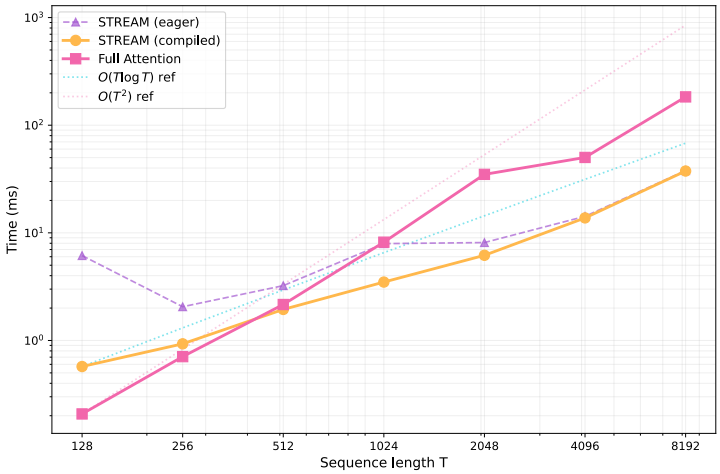

Figure 1: Performance of the STREAM algorithm compared to full attention, on Qwen 3 0.6B, with $k = 2$.

The idea is based on a hierarchical search, similar to a binary search, which identifies the most relevant key blocks for each query block without having to evaluate the entire attention matrix. The algorithm proceeds as follows:

- **Initialization:** The sequence is first divided into query blocks of size b_q and key blocks of size b_k. STREAM starts by splitting the entire key pattern into k branches of equal size.
- **Pruning:** At each iteration, the algorithm identifies which valid branches contain the highest attention scores. It then discards the least relevant regions.
- **Dichotomy:** With each new iteration, the search space is halved. STREAM recursively subdivides the remaining promising branches, continuing this hierarchical refinement until converging to a final set of k key blocks (of dimension $b_q \times b_k$) that exhibit the strongest attention links with each query block.

The algorithm has three hyper-parameters:

- b_q: the size of the query block.
- b_k: the size of the key block.
- k: the sparsity constant, which controls the overall level of pruning.

One of the advantages of STREAM is that these parameters can be adjusted to match linguistic units. For example, setting $b_q = b_k = 32$ allows for an analysis at the sentence level, while blocks set to 28 are more suitable for paragraph-level analysis. The global sparsity level is defined by the constant $k \in \left[1, \left\lfloor \frac{T}{b_k} \right\rfloor\right]$. For a given input, the STREAM method generates sparse masks for each attention head across all layers, following three steps:

1. **Mask calculation:** The algorithm calculates the hierarchical sparse attention masks using the specified parameters (b_q, b_k, k).
2. **Application:** These masks are applied to the complete attention patterns in order to isolate only the most relevant connections.
3. **Sparsity Optimization:** A binary search is performed on the constant k. The objective is to find the lowest possible value of k (favoring parsimony) that allows the model to generate the same next two tokens as the original unpruned model.

The criterion of two tokens generated consecutively in the same manner is used as a proxy to evaluate perplexity and maintain model performance (this will be discussed later). This ensures that the attention pattern accurately captures the relevant behaviors of the network. However, we avoid doing it on the first layers because it destroys the model's performance.

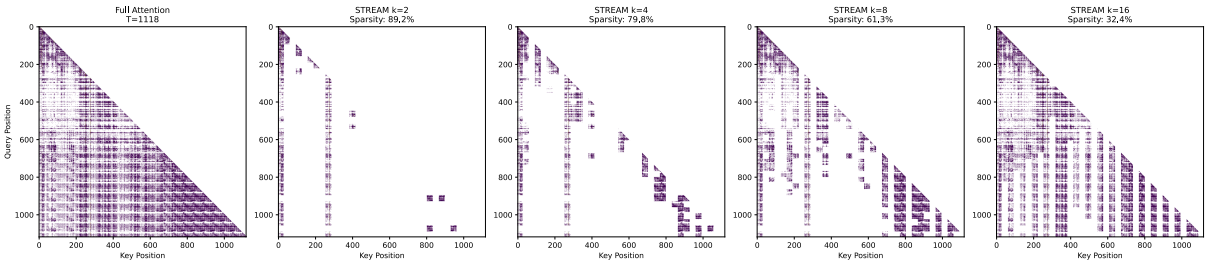

Figure 2: STREAM attention patterns with Qwen 3 0.6B on the receiver head L21 H10, with $b = 32$.

4 Case Studies and Validation

4.1 Similarity with full attention

To validate their results, the authors used the cosine similarity metric, along with a recall and top 1 output token match. Even if these metrics can not assess the performance on tasks like MMLU, they serve as a proxy.

In general, the cosine similarity stays high, but the top-k best tokens drop significantly (Figure 3). In practice, the authors mentionned that generating an answer with sparse attention only recovered the first two tokens, and diverged later, but it was enough to reproduce the results of Thought Anchors (Bogdan et al., 2025), and the Needle in a Haystack (Nelson et al., 2024) papers.

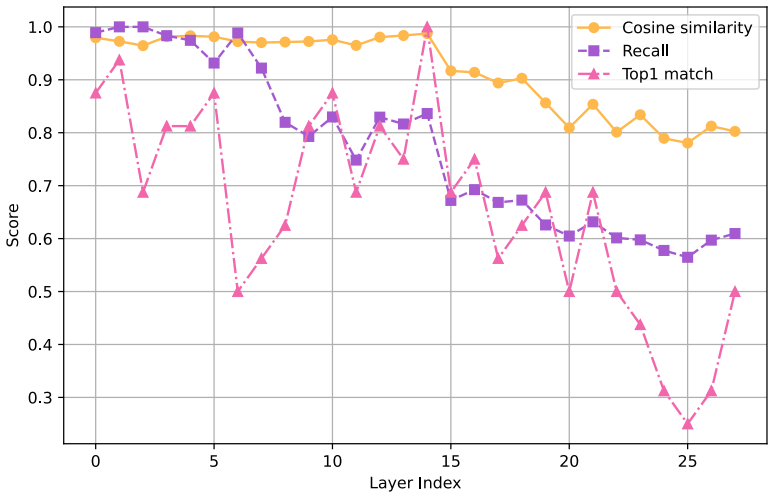

Figure 3: Similarity using STREAM on Qwen 3 0.6B, with a math question, a full length of 1153, $k = 8, b = 32$.

4.2 Replication of *Thought Anchors* and *The Needle in a Haystack*

The authors tested STREAM on the interpretability tasks Thought Anchors (Bogdan et al., 2025) and the Needle in a Haystack (Nelson et al., 2024).

The Thought Anchors' experiment aims to find the specific steps in reasoning (Chain-of-Thought) that exert a significant influence on the rest of the text generation. On this task, with lengths of 10 000 tokens, STREAM was able to recover most of the anchor heads, with only 1-3% of the attention. Making the process much more efficient.

The evaluation of the STREAM algorithm on the Needle in a Haystack task demonstrates an ability to reduce computational complexity without signal loss. By analyzing contexts of up to 20,000 tokens, the authors observe that even when aggressively pruning between 90 and 96% of attention connections, the generated masks remain structured. The path leading to the sought information remains intact and observable, proving that STREAM manages to preserve the essence of the signal on large scales.

However, the depth analysis reveals a structural limitation compared to classical models. Unlike the usual "U" tracking curve (where models perform well at the beginning and end of the text), STREAM is very robust at retrieving information at the beginning of the context, but its performance is poor toward the end for texts longer than 8,000 tokens (as soon as the pruning factor $s \geq 0.1$). This phenomenon is explained by the lower triangular nature of causal attention: the further we advance in the text, the more key blocks there are to compare. With a constant sparsity parameter k, the algorithm ends up pruning the last positions too much.

5 Experiments: Adversarial attacks

Even if STREAM retains 95%+ of important attention, the dropped 5% could be catastrophically important. We find a small set of adversarial tokens (3-4) that, prepended to a yes/no question, cause the model to answer differently under sparse vs full attention — while both see the same prompt.

The adversarial tokens create attention patterns where the blocks STREAM drops happen to carry critical information for the answer.

One way to generate adversarial inputs would be to use a gradient-based algorithm, like the Greedy Coordinate Gradient attack (Zou et al., 2023). However, as LLMs are brittle, it is often enough to test tokens at random.

Find adversarial tokens

Given the context $c \in V^{n_{\text{context}}}$, n_{adv} adversarial tokens $x \in V^{n_{\text{adv}}}$ initialized at random, the prompt containing the Yes/No question $p \in V^{n_{\text{question}}}$, the set of correct answers $a \subset V$, for instance, if the correct answer is "Yes", a can be $\{\text{Yes}, \text{Yes} , \text{yes}\}$, the attack sparsity k_{attack}, and a set of K test sparsities:

$y \leftarrow \text{LLM}(c \oplus x \oplus p)$
if $y \notin a$ **then**
  Exit "LLM can't answer with full attention"
for T **steps:**
  $y \leftarrow \text{SparseLLM}(k_{\text{attack}}, c \oplus x \oplus p)$
  if $y \notin a$ **then**
    Break
  $x \leftarrow \text{Uniform}(V)^{n_{\text{adv}}}$
for $k \in K$**:**
  $y_k \leftarrow \text{SparseLLM}(k, c \oplus x \oplus p)$
return $x, y, \{y_k, \forall k \in K\}$

If the attack succeeds, y is in a, and for all $k \leq k_{\text{attack}}$, y_k is also in a, so we verify that the result is not only noise, but an artefact of sparsity. We also add a context before the prompt, to increase the length

and the effect of sparsity. In our experiments, we took the `Qwen/Qwen3-0.6B` model, with $k_{\text{attack}} = 4, b = 8, n_{\text{adv}} = 3$.

Table 1: Results of the attacks. The vanilla probabilities use $c \oplus p$ and full attention, sparse probabilities use $c \oplus p$ and sparse attention with k_{adv}, other values were computed with $c \oplus x \oplus p$, with the correct tokens' probability at the left, and the wrong tokens' probability at the right. Bold values indicate a flip in the answer.

	Vanilla probs	Sparse probs	$k_1 = 8$	$k_{\text{adv}} = 4$	$k_2 = 2$	Adv tokens
Is the sun a star?	0.45	0.50	0.53/ 0.01	**0.00/ 0.01**	**0.010/ 0.016**	Dungeon actᴛ 磔
Is Rome in Belgium?	0.82	**0.005**				
Is 100 an even number?	0.48	0.003	0.52/ 0.001	**0.002/ 0.003**	**0.000/ 0.001**	.hp kü _dot
Is fire hot?	0.25	0.09	0.09/ 0.02	**0.001/ 0.004**	**0.000/ 0.001**	Note most ق

As we can see in Table 1, simple sentences can be attacked with random search. In practice, we had approximately 80% success rate, but more importantly, some sentences don't even need adversarial tokens. Asked with the question "Is Rome in Belgium?", the LLM with sparse attention already gave the wrong answer.

This small experiment shows the brittleness of the tested LLMs, and the STREAM algorithm. Even if the STREAM algorithm retreive 95% of the information measured with KL divergence or cosine similarity, the lost information can completely change the behaviour of the model, making the STREAM algorithm unreliable.

6 Conclusion

The STREAM algorithm effectively reduces attention costs drastically, and enables some interpretability analysis, like the Thought Anchors, or the Needle in a Haystack. However, even if it keeps most of the information, the dropped attention can completely change the behaviour of the model, wich some extreme cases where simple answers can flip from "Yes" to "No". Thus, it can not be used for use cases where reliability is mandatory, like safety analysis, attack prevention, or backdoor detection.

References

Bogdan, P. C., Macar, U., Nanda, N., and Conmy, A. *Thought Anchors: Which LLM Reasoning Steps Matter?*, 2025. https://arxiv.org/abs/2506.19143

Lee, H., Park, G., Lee, Y., Suh, J., Kim, J., Jeong, W., Kim, B., Lee, H., Jeon, M., and Hwang, S. J. *A Training-free Sub-quadratic Cost Transformer Model Serving Framework With Hierarchically Pruned Attention*, 2025. https://arxiv.org/abs/2406.09827

Nelson, E., Kollias, G., Das, P., Chaudhury, S., and Dan, S. *Needle in the Haystack for Memory Based Large Language Models*, 2024. https://arxiv.org/abs/2407.01437

Rosser, J., García, J. L. R., Penha, G., Palla, K., and Bouchard, H. *Stream: Scaling up Mechanistic Interpretability to Long Context in LLMs via Sparse Attention*, 2025. http://arxiv.org/abs/2510.19875 v2

Templeton, A., and others. Towards Monosemanticity: Decomposing Language Models With Sparse Autoencoders. *Anthropic*, 2023.

Zou, A., Wang, Z., Carlini, N., Nasr, M., Kolter, J. Z., and Fredrikson, M. *Universal and Transferable Adversarial Attacks on Aligned Language Models*, 2023. http://arxiv.org/abs/2307.15043v2