SpecAttn: Speculating Sparse Attention

Harsh Shah
Machine Learning Department
Carnegie Mellon University
hshah2@cs.cmu.edu

Abstract

Large Language Models (LLMs) face significant computational bottlenecks during inference due to the quadratic complexity of self-attention mechanisms, particularly as context lengths increase. We introduce SpecAttn, a novel training-free approach that seamlessly integrates with existing speculative decoding techniques to enable efficient sparse attention in pre-trained transformers. Our key insight is to exploit the attention weights already computed by the draft model during speculative decoding to identify important tokens for the target model, eliminating redundant computation while maintaining output quality. SpecAttn employs three core techniques: KL divergence-based layer alignment between draft and target models, a GPU-optimized sorting-free algorithm for top-p token selection from draft attention patterns, and dynamic key-value cache pruning guided by these predictions. By leveraging the computational work already performed in standard speculative decoding pipelines, SpecAttn achieves over 75% reduction in key-value cache accesses with a mere 15.29% increase in perplexity on the PG-19 dataset, significantly outperforming existing sparse attention methods. Our approach demonstrates that speculative execution can be enhanced to provide approximate verification without significant performance degradation.

1 Introduction

Transformer architectures have become the de-facto standard for sequence modeling across natural language processing tasks, owing to their ability to capture long-range dependencies via self-attentionVaswani et al. [2023]. However, the self-attention mechanism incurs time and memory costs that scale quadratically with sequence length, severely constraining model throughput and context window sizes in practice. To alleviate these inference bottlenecks, system-level engines such as vLLM employ optimized batching, GPU utilization, and key–value cache management to deliver 1.8–2.7× speedups over standard frameworks, yet they continue to execute full dense attention over the entire context Kwon et al. [2023].

Complementary algorithmic solutions introduce sparse attention patterns to reduce pairwise token interactions. The Longformer work Beltagy et al. [2020] adopts sliding-window and global attention spans to achieve linear complexity with respect to sequence length while BigBird Zaheer et al. [2021] combines local, random, and global attention heads to extend effective context lengths by up to eight times, with theoretical guarantees of universal approximation.

Despite their efficiency gains, these sparse architectures depend on predetermined attention patterns that are agnostic to individual input content and require model retraining or fine-tuning to adapt to new domains, limiting their flexibility for dynamic inference scenarios.

An alternative paradigm, speculative decoding, uses a lightweight draft model to generate candidate token sequences in parallel, which a larger verifier model then validates Leviathan et al. [2023]. This

draft-and-verify approach achieves throughput improvements on large transformer models without altering output distributions or requiring weight updates.

While prior research has treated speculative decoding and sparse attention as orthogonal optimization strategies, the attention patterns computed by draft models during speculative execution contain rich information about token importance that can be leveraged for dynamic, content-aware key-value cache pruning without much additional computational overhead.

We summarize our contributions below:

- We provide a method to map draft model's layers to verifier model layers so that sparse attention can be speculated, using KL divergence as the similarity metric between attention distributions.

- We introduce a Triton kernel that identifies top-p nucleus tokens from the draft model's attention distribution without requiring sorting operations, enabling efficient GPU utilization by eliminating the computational overhead of explicit sorting.

2 Related Works

System-Level Optimization. Recent research has developed numerous techniques to accelerate LLM inference without sacrificing much accuracy. Microsoft's DeepSpeed library Rasley et al. [2020] introduces kernel fusion and customized GPU kernels to speed up inference, achieving significant speedups on models like GPT-2 and BERT compared to PyTorch baseline inference. However, kernel fusion still computes the full attention matrix, making its latency grow steeply with sequence length. High-throughput KV-cache management in frameworks such as vLLM Kwon et al. [2023] and optimized servers like HuggingFace's Text Generation Inference harness smart batching and prefix caching to maximize GPU utilization and reuse computed states. FlashAttention Dao et al. [2022], Dao [2023], Shah et al. [2024] leverages on-chip tiling and softmax fusion to reduce memory traffic and yields substantial latency reductions on long sequences. These methods deliver significant speedups on popular LLMs with minimal changes to existing models, but they still execute the full $O(n^2)$ attention: every query–key pair is computed regardless of its eventual impact on output quality.

Sparse Attention Architectures. Parallel to system-level advances, sparse-attention architectures and dynamic, content-dependent pruning approaches have aimed to break the quadratic barrier by only attending to a subset of tokens. Static sparse models such as Longformer Beltagy et al. [2020] and BigBird Zaheer et al. [2021] impose hand-crafted patterns—sliding windows, global tokens, or random blocks—to achieve linear or near-linear complexity, but require retraining with those fixed masks and cannot adapt to the unique needs of each input. More recent inference-time methods such as MInference Jiang et al. [2024] and SpargeAttn Zhang et al. [2025] select top-k keys or apply two-stage filtering to drop low-importance attention links on the fly, offering faster prompt processing without retraining. Additional approaches include Quest Tang et al. [2024], which uses query-aware KV cache page selection, StreamingLLM Xiao et al. [2023], which leverages attention sinks for infinite sequence length generalization, and Twilight Lin et al. [2025], which applies adaptive top-p pruning for hierarchical attention sparsity. However, these semi-dynamic schemes still rely on predetermined head patterns or incur extra prediction overhead, and their benefits diminish on shorter contexts or during the token-by-token generation phase.

Speculative Decoding. Speculative decoding Leviathan et al. [2023], Chen et al. [2023] offers an orthogonal speedup by pairing a small "draft" model with the full-scale LLM: the draft proposes tokens in batches that the verifier then checks, reducing the number of verifier model invocations. While this approach avoids retraining the main model, it does not change its internal attention cost and depends heavily on the draft's accuracy. Previous works have addressed speculative decoding and sparse attention as separate acceleration techniques, yet the attention distributions inherent to draft model computation provide a natural signal for identifying salient tokens. SpecAttn is motivated by these gaps: it unites speculative decoding's training-free paradigm with fully dynamic, content-aware sparse attention. By exploiting the attention weights already computed by the draft model to score and select the most informative tokens at each generation step, SpecAttn prunes the KV-cache significantly without any model modifications, preserving output fidelity while delivering end-to-end latency improvements.

3 Method

In this section, we present the methodology behind SpecAttn, our novel approach for accelerating inference in LLMs through dynamic sparse attention. We first introduce the problem formulation and then describe our framework in detail.

3.1 Problem Formulation

3.1.1 Sparse Attention Formulation

Let M_v denote the verifier model with m layers and M_d denote the draft model with n layers. During the decoding phase of language model inference, we have the query vector $Q \in \mathbb{R}^{1 \times d}$ and the key-value cache $K, V \in \mathbb{R}^{L \times d}$, where d is the head dimension and L is the context length.

In standard dense attention, the output is computed as:

$$O = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) V = WV \tag{1}$$

where $W = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) \in \mathbb{R}^{1 \times L}$ represents the normalized attention weights.

The computational complexity of this operation is $O(L^2 d)$, which becomes prohibitively expensive for long sequences. To reduce this complexity, we introduce sparse attention by selecting only a subset of tokens to attend to.

3.1.2 Sparse Attention with Token Selection

Let $\mathcal{I}$ be the set of selected token indices. We define the sparse attention operation as:

$$\hat{O} = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) \Lambda_{\mathcal{I}} V = W \Lambda_{\mathcal{I}} V \tag{2}$$

where $\Lambda_{\mathcal{I}} \in \mathbb{R}^{L \times L}$ is a diagonal mask matrix defined as:

$$\Lambda_{\mathcal{I}}[i, j] = \begin{cases} 1 & \text{if } i = j \text{ and } i \in \mathcal{I} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

The theoretical error bounds and optimization objectives for this formulation are detailed in Appendix A.

3.2 SpecAttn Framework Overview

Our SpecAttn framework leverages the complementary strengths of a lightweight draft model and a high-capacity verifier model in a cooperative inference pipeline. The core insight is that the attention patterns in the draft model often provide a good approximation of the important tokens that the verifier model should attend to. Figure 1 illustrates the overall architecture of our approach.

The SpecAttn framework consists of three key steps:

1. **Layer Mapping**: Establish correspondence between draft and verifier model layers using KL divergence similarity between attention distributions.
2. **Token Selection**: Identify important tokens using sorting-free top-p nucleus selection from draft model attention patterns.
3. **Sparse Attention Computation**: Perform sparse attention computation by attending only to selected tokens in the previous step.

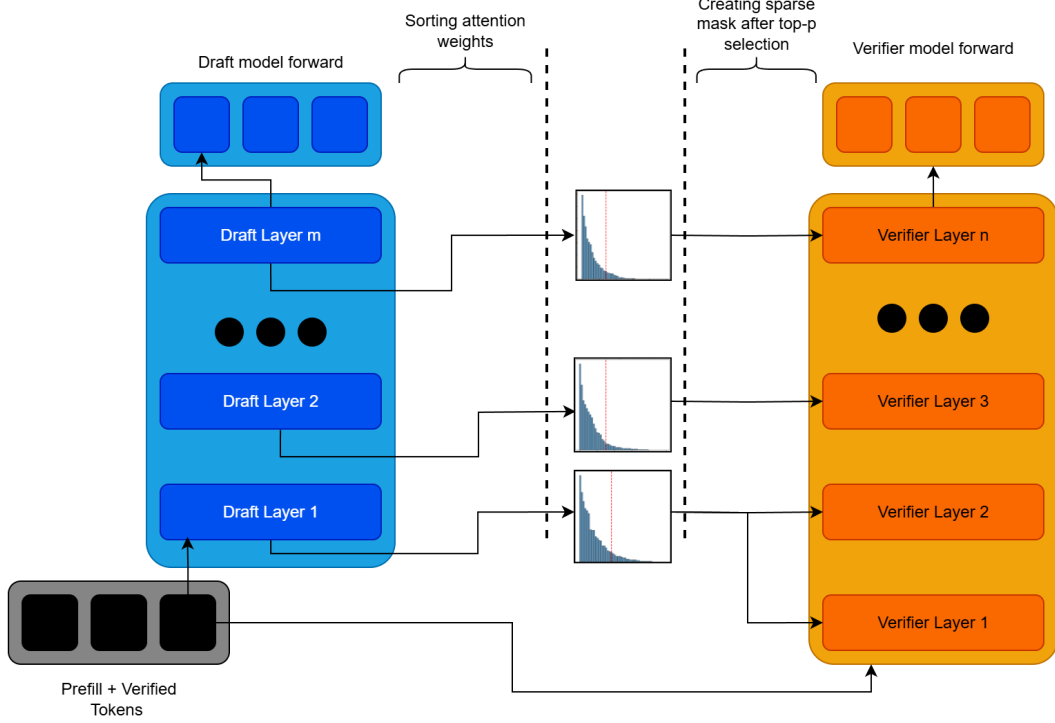The overall workflow of SpecAttn is summarized in Algorithm 3.2

3

Figure 1: SpecAttn framework. Layer specific tokens are dynamically selected during runtime from draft model to prune KV cache of target model. Notice that a layer draft model can be mapped to multiple layers in verifier model. Also, the sorting described here is just for illustration, the implementation described in this paper performs the selection without sorting, Algo 2

3.3 Layer Mapping via KL Divergence

A critical challenge in our approach is establishing an optimal mapping between the layers of the draft model and the verifier model, especially when these models have different depths. We formulate this as finding the layer in the draft model that has the most similar attention distribution to each layer in the verifier model. Note that this mapping is formed offline using a representative dataset (wikitext Merity et al. [2016]) and does not change during runtime.

Let $A_i^d \in \mathbb{R}^L$ and $A_j^v \in \mathbb{R}^L$ represent the attention distributions from layer i of the draft model and layer j of the verifier model, respectively. We define the similarity between these layers using the KL divergence:

$$S_{i,j} = -D_{KL}(A_j^v || A_i^d) = -\sum_{k=1}^{L} A_j^v[k] \log \frac{A_j^v[k]}{A_i^d[k]} \quad (4)$$

For each verifier layer j, we find the best matching draft layer while keeping the index chosen for draft layer monotonically increasing. The intuition behind monotonic mapping is that the attention patterns are similar for different models as one moves up the models' layers. The mapping problem is a modification to dynamic time warping and can be efficiently solved by dynamic programming techniques. Pseudo code and more details regarding the mapping can be found in appendix B.

3.4 Sorting-Free Top-p Nucleus Selection

Once we have established the layer mapping, we leverage the draft model's attention distributions to determine which tokens the verifier model should attend to. Traditional top-p selection requires

4

Algorithm 1 SpecAttn: Speculative Sparse Attention Framework

1: **Input:** Input sequence $\mathbf{x} \in \mathbb{R}^L$, draft model M_d, verifier model M_v, lookahead γ, threshold p
2: **Output:** Generated sequence $\mathbf{y}$
3: **Initialization:**
4: Initialize KV caches: $\mathcal{K}_d, \mathcal{V}_d$ for draft, $\mathcal{K}_v, \mathcal{V}_v$ for verifier
5: Compute layer mapping $f : [m] \to [n]$ via $\arg\max_i$ KL-sim(A_j^v, A_i^d)
6: Prefill both models: $M_d(\mathbf{x}) \to \mathcal{K}_d, \mathcal{V}_d$; $M_v(\mathbf{x}) \to \mathcal{K}_v, \mathcal{V}_v$
7: $\mathbf{y} \leftarrow \mathbf{x}, t \leftarrow |\mathbf{x}|$ $\triangleright$ Initialize output and position counter
8: **while** $t < T_{\max}$ and not EOS **do**
9:      **// Speculative Generation Phase**
10:      $\mathcal{S} \leftarrow \{\}, \mathcal{A} \leftarrow \{\}$ $\triangleright$ Initialize buffers
11:      **for** $s = 1$ to γ **do** $\triangleright$ Generate γ speculative tokens
12:          $\mathbf{z}_s \leftarrow \mathbf{y}_{t-1}$ if $s=1$ else $\mathcal{S}_{s-1}$ $\triangleright$ Use last accepted token
13:          $\mathbf{o}_s^d, A_{1:n,s}^d \leftarrow M_d(\mathbf{z}_s, \text{cache} = \mathcal{K}_d, \mathcal{V}_d)$
14:          $\mathcal{S} \leftarrow \mathcal{S} \cup \{\arg\max(\mathbf{o}_s^d)\}$ $\triangleright$ Store draft token
15:          $\mathcal{A} \leftarrow \mathcal{A} \cup \{A_{1:n,s}^d\}$ $\triangleright$ Store attention patterns
16:          Update $\mathcal{K}_d, \mathcal{V}_d$ with $\mathbf{S}_{-1}$ $\triangleright$ Update draft model's KV cache
17:      **end for**
18:      **// Sparse Attention Mask Creation**
19:      **for** $j = 1$ to m **do** $\triangleright$ For each verifier layer
20:          $i^* \leftarrow f(j)$ $\triangleright$ Get corresponding draft layer index
21:          $\mathcal{T}_j \leftarrow \text{SortingFreeNucleus}(\{\mathcal{A}_{i^*}^d\}_{d=1}^\gamma, p)$ $\triangleright$ Algorithm 2
22:          $\mathcal{M}_j \leftarrow \text{CreateSparseMask}(\mathcal{T}_j, |\mathcal{K}_v|)$ $\triangleright$ Binary attention mask
23:      **end for**
24:      **// Verification with Sparse Attention**
25:      $\mathbf{H}^v, \mathbf{O}^v \leftarrow M_v(\mathcal{S}, \text{masks} = \{\mathcal{M}_j\}_{j=1}^m, \text{cache} = \mathcal{K}_v, \mathcal{V}_v)$
26:      $\hat{\mathbf{y}}_{1:\gamma+1} \leftarrow \arg\max(\mathbf{O}^v)$ $\triangleright$ Verifier predictions
27:      **// Token Verification & Acceptance**
28:      $n_{acc} \leftarrow \text{CheckAcceptance}(\mathcal{S}_{1:\gamma}, \hat{\mathbf{y}}_{1:\gamma})$ $\triangleright$ Count accepted tokens
29:      $\mathbf{y}_{t+1:t+n_{acc}} \leftarrow \mathcal{S}_{1:n_{acc}}$ $\triangleright$ Accept verified tokens
30:      $t \leftarrow t + n_{acc} + 1$ $\triangleright$ Update position counter
31:      **// Cache Management**
32:      Update $\mathcal{K}_d, \mathcal{V}_d$ with accepted tokens $\mathbf{y}_{t-n_{acc}:t}$
33:      Update $\mathcal{K}_v, \mathcal{V}_v$ with accepted tokens $\mathbf{y}_{t-n_{acc}:t}$
34: **end while**
35: **return** $\mathbf{y}$

---

sorting the attention weights, which can be computationally expensive on GPUs. We propose a sorting-free algorithm for efficient nucleus identification inspired from work in Lin et al. [2025].

This binary search approach finds the optimal threshold without explicit sorting, making it more GPU-friendly. For each layer j in the verifier model, we:

1. Extract attention weights $\mathbf{a}_{f(j)} \in \mathbb{R}^L$ from the corresponding draft model layer $f(j)$.

2. Apply Algorithm 2 [1] to find the minimal subset of tokens $\mathcal{T}_j \subseteq \{1, \ldots, L\}$ such that:
   $\sum_{k \in \mathcal{T}_j} a_{f(j),k} \geq p$ where $p \in (0, 1]$ is a predefined threshold (e.g., $p = 0.95$).

This approach corresponds to performing top-p (nucleus) sampling over the attention weights, ensuring that we retain tokens accounting for at least p fraction of the total attention mass. The parameter p allows us to control the trade-off between computational efficiency and output quality.

3.5 Sparse Attention Computation

To compute sparse attention with the selected tokens, we employ the sparse attention kernel from Flashinfer Ye et al. [2025]. The generated mask containing selected tokens is converted to compressed

---

[1] the implementation in this work uses a fixed number of iterations (i.e., 10)

Algorithm 2 Sorting-Free Top-p Nucleus Selection

1: **procedure** SORTINGFREENUCLEUS($\mathcal{A}_{layer} \in \{\mathbb{R}^{1 \times L}\}_{\gamma}, p \in (0, 1]$)
2:     $\mathcal{T} \leftarrow \{\}$ ▷ Initialize combined token set
3:     **for** $s = 1$ to γ **do**   ▷ Process each speculative step
4:         $\mathbf{a}_s \leftarrow \mathcal{A}_{layer,s}$ ▷ Get attention weights for step s
5:         $M_{target} \leftarrow p \cdot \sum_{i=1}^{L} a_{s,i}$ ▷ Target attention mass
6:         $\theta_{high} \leftarrow \max(\mathbf{a}_s), \theta_{low} \leftarrow 0$ ▷ Binary search bounds
7:         $\epsilon \leftarrow 10^{-6}$ ▷ Convergence tolerance
8:         **while** $\theta_{high} - \theta_{low} > \epsilon$ **do**
9:             $\theta_{mid} \leftarrow \frac{\theta_{high} + \theta_{low}}{2}$
10:             $M_{current} \leftarrow \sum_{i:a_{s,i} \geq \theta_{mid}} a_{s,i}$ ▷ Mass above threshold
11:             **if** $M_{current} < M_{target}$ **then**
12:                 $\theta_{high} \leftarrow \theta_{mid}$ ▷ Lower threshold
13:             **else**
14:                 $\theta_{low} \leftarrow \theta_{mid}$ ▷ Raise threshold
15:             **end if**
16:         **end while**
17:         $\mathcal{T}_s \leftarrow \{i : a_{s,i} \geq \theta_{mid}\}$ ▷ Selected tokens for step s
18:         $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{T}_s$ ▷ Union with global token set
19:     **end for**
20:     **return** $\mathcal{T}$ ▷ Return combined token indices
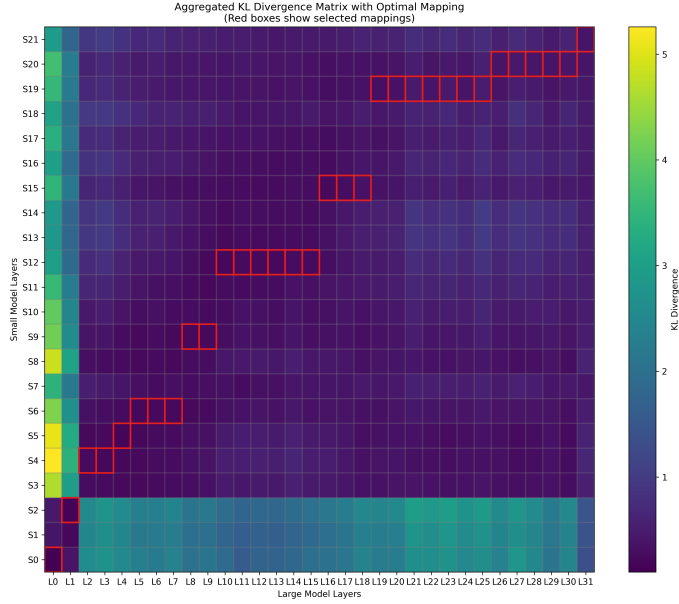21: **end procedure**

Figure 2: Heatmap of KL divergence between layers of **TinyLlama-1.1B** as small model (draft model) and **Llama-2-7b-hf** as large model (verifier model). Red boxes denote the draft layer selected for the corresponding verifier layer.

sparse row (CSR) format, which introduces additional latency during generation, before being used for sparse attention computation.

4 Experiments

We have conducted all our experiments using a single NVIDIA RTX 4090 GPU, with 24GB VRAM. For all experiments in this section, we use **TinyLlama-1.1B**[2] as draft model and **Llama-2-7b-hf**[3] as the target model. The mapping for these models obtained by method described in 3.3 is shown in Fig 2. For analsyis of perplexity, we use PG-19 dataset Rae et al. [2019] truncated to 2048 tokens, out of which 10% are used for prefill and other 90% are used for evaluating decoding process, simulating long generation scenarios. We also measure KV computation reduction and end-to-end latency on LongBench Bai et al. [2024] task gov_report to compare SpecAttn with full attention speculative decoding (with the similar prefill and decoding setup as for perplexity analysis).

4.1 Perplexity Evaluation

We evaluate the perplexity of different sparse attention methods to assess their impact on generation quality. Table 1 shows the comparison between our method and existing sparse attention techniques. Note that the first 2 layers use full attention for all methods due to diffused attention found in initial layers (described in Tang et al. [2024]). Further the chunk size of SpecAttn is set to 16 for fair comparison with Quest (Tang et al. [2024]). It can be observed that SpecAttn (p=0.95) reduces KV cache loading by 78.4% while increasing perplexity by mere 0.984 (15.29% relative increase), which significantly improves upon other sparse attention methods.

Table 1: Performance Comparison of Sparse Attention Methods

Method	Perplexity	Perp. Diff.	Rel. Increase	KV Reduction
Full Attention	6.435	-	-	-
StreamingLLM (Xiao et al. [2023])	186.242	+179.807	+2794.32%	77.4%
Quest (Tang et al. [2024])	7.823	+1.389	+21.58%	77.4%
SpecAttn (p=0.95)	7.419	+0.984	+15.29%	**78.4%**
SpecAttn (p=0.97)	6.720	+0.285	+4.43%	68.8%
SpecAttn (p=0.99)	**6.471**	**+0.036**	**+0.56%**	44.3%

4.2 Speedup Analysis

We show the speedup achieved by using sorting-free triton kernel (as described in 2) when compared to sorting-based algorithm, in Fig 3. It can be observed that there is at least 4x speedup till KV cache size 8192 by uing sorting-free trtion kernel. For computing sparse attention, we use Flashinfer's Ye et al. [2025] BlockSparseAttention [4]. In Fig 4, we show the speedup obtained when using p=0.97 as compared to p=1.0 (full attention). It shows an increasing trend of speedup as prompt length increases, with more than 4x speedup for prompt length 2048.

We also evaluated SpecAttn using end-to-end latency comparison with full attention speculative decoding (Table 2). The end-to-end latency seems to be higher for SpecAttn, and this can be attributed to high mask generation time (Algo 2) which is compensated through sparsity in KV cache. However, the trend in Fig 4 gives promising direction of increasing the context length for the experiments, since that would lead to further reduction of latency in attention computation, compensating for the mask generation time.

Table 2: Throughput experiments

Method	Tokens/sec ($\uparrow$)	KV Reduction ($\uparrow$)
No Spec. Decoding (flashattn)	42.00	-
Spec. Decoding (Full Attention)	**68.26**	-
SpecAttn (p=0.97)	59.95	**71.89%**

---

[2]https://huggingface.co/TinyLlama/TinyLlama-1.1B-intermediate-step-1431k-3T

[3]https://huggingface.co/meta-llama/Llama-2-7b-hf

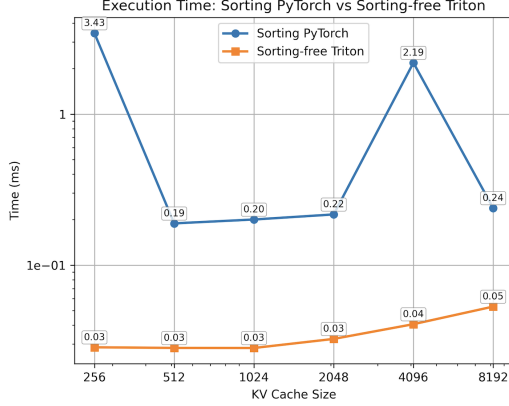[4]https://docs.flashinfer.ai/api/sparse.html

Figure 3: Graph of time taken for mask generation when comparing sorting in pytorch with sorting-free algorithm as depicted in Algo 2
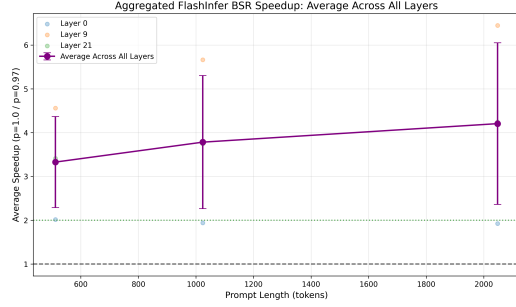
Figure 4: Speedup achieved in computing attention through FlashInfer's BlockSparseAttention with p=0.97 as compared to p=1.0 (i.e., full attention)

5 Conclusion

In this work, we introduced SpecAttn, a training-free framework that fuses speculative decoding with dynamic sparse attention to drastically reduce the key-value (KV) cache usage in LLM inference. By mapping layers using KL divergence similarity and employing a sorting-free top-p token selection mechanism at each decoding step, SpecAttn achieves significant KV cache reductions and maintains competitive perplexity compared to existing sparse attention methods.

Our empirical evaluations demonstrate that SpecAttn can reduce KV cache access by up to 78% while maintaining reasonable perplexity degradation (15.29% relative increase for p=0.95). The method excels at balancing efficiency and quality, outperforming existing methods like Quest at similar sparsity levels through context-aware and dynamic pruning.

6 Limitations and Future Work

Several promising directions remain for future investigation. While we focused on KL divergence for layer mapping, exploring alternative similarity metrics (jaccard similarity, other distribution distances, etc.) could potentially improve the quality of layer correspondences between draft and verifier models. Additionally, extending our evaluation to much longer contexts (10K+ tokens) would provide better understanding of SpecAttn's scaling properties and effectiveness in truly long-context scenarios where the quadratic attention cost becomes more prohibitive. Finally, integrating SpecAttn into production serving frameworks like vLLM to leverage PagedAttention and emerging dynamic token caching (DTC) capabilities would enable comprehensive benchmarking and real-world deployment validation.

References

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL `https://arxiv.org/abs/1706.03762`.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020. URL `https://arxiv.org/abs/2004.05150`.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences, 2021. URL `https://arxiv.org/abs/2007.14062`.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023. URL `https://arxiv.org/abs/2211.17192`.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 3505–3506, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3406703. URL `https://doi.org/10.1145/3394486.3406703`.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. URL `https://arxiv.org/abs/2205.14135`.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning, 2023. URL `https://arxiv.org/abs/2307.08691`.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision, 2024. URL `https://arxiv.org/abs/2407.08608`.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention, 2024. URL `https://arxiv.org/abs/2407.02490`.

Jintao Zhang, Chendong Xiang, Haofeng Huang, Jia Wei, Haocheng Xi, Jun Zhu, and Jianfei Chen. Spargeattn: Accurate sparse attention accelerating any model inference, 2025. URL `https://arxiv.org/abs/2502.18137`.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference, 2024.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv*, 2023.

Chaofan Lin, Jiaming Tang, Shuo Yang, Hanshuo Wang, Tian Tang, Boyu Tian, Ion Stoica, Song Han, and Mingyu Gao. Twilight: Adaptive attention sparsity with hierarchical top-p pruning, 2025. URL `https://arxiv.org/abs/2502.02770`.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023. URL `https://arxiv.org/abs/2302.01318`.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. Flashinfer: Efficient and customizable attention engine for llm inference serving. *arXiv preprint arXiv:2501.01005*, 2025. URL `https://arxiv.org/abs/2501.01005`.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, Chloe Hillier, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint*, 2019. URL `https://arxiv.org/abs/1911.05507`.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL `https://aclanthology.org/2024.acl-long.172`.

A   Error Bounds and Optimization Objective

In this appendix, we provide the theoretical foundation for our sparse attention formulation introduced in the main paper. We establish error bounds for the sparse attention approximation and derive the optimization objective that motivates our top-p token selection strategy.

A.1   Problem Context and Motivation

Large language models compute attention over increasingly long sequences, leading to quadratic computational complexity $O(L^2 d)$ where L is the sequence length and d is the head dimension. Our sparse attention framework addresses this by selecting only a subset of tokens to attend to, reducing both computational cost and memory requirements.

Given the standard dense attention output $O = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) V = WV$ and our sparse approximation $\hat{O} = W\Lambda_{\mathcal{I}} V$ where $\mathcal{I}$ represents selected token indices, we seek to minimize the approximation error while maximizing computational savings.

A.2   Error Bound and Optimization Objective

The output error can be bounded as follows:

$$\mathcal{L} = \|O - \hat{O}\| = \|W(\Lambda_{\mathcal{I}} - I_{L \times L})V\| \tag{5}$$
$$\leq \|W(\Lambda_{\mathcal{I}} - I_{L \times L})\| \cdot \|V\| \tag{6}$$

Our objective becomes minimizing $\|W(\Lambda_{\mathcal{I}} - I_{L \times L})\| = 1 - \sum_{i \in \mathcal{I}} W[i]$. This means we want to select tokens that maximize the sum of attention weights, preserving the most important attention mass.

A.3   From Top-k to Top-p Sparse Attention

Traditional approaches to sparse attention use a fixed budget B (top-k) to select tokens:

$$\mathcal{I} = \arg\max_{\mathcal{I}} \sum_{i=1}^{L} W\Lambda_{\mathcal{I}} \quad \text{s.t.} \quad |\mathcal{I}| = B \tag{7}$$

However, a major limitation of the top-k approach is that it cannot adapt to different attention weight distributions. Attention patterns can vary significantly between flat (diffuse) and peaked (focused) distributions. A fixed budget B would either be inefficient for peaked distributions or insufficient for flat distributions.

To address this issue, we propose using a top-p (nucleus) approach for token selection:

$$\mathcal{I} = \arg\min_{\mathcal{I}} |\mathcal{I}| \quad \text{s.t.} \quad \sum_{i \in \mathcal{I}} W[i] \geq p \tag{8}$$

where $p \in (0, 1]$ is a threshold parameter. This formulation dynamically adjusts the number of tokens based on the attention distribution, selecting just enough tokens to capture p fraction of the total attention mass.

The top-p approach provides a theoretical error bound of $(1 - p) \cdot \|V\|$, making it possible to control the trade-off between computational efficiency and output quality by adjusting the parameter p.

B   Layer Mapping Algorithm

The layer mapping problem can be formulated as finding an optimal monotonic alignment between draft model layers and verifier model layers. Given a similarity matrix $S \in \mathbb{R}^{m \times n}$ where $S[i, j]$ represents the compatibility score between draft layer i and verifier layer j, we seek a mapping function $f : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, m\}$ that maximizes the total alignment score $\sum_{j=1}^{n} S[f(j), j]$
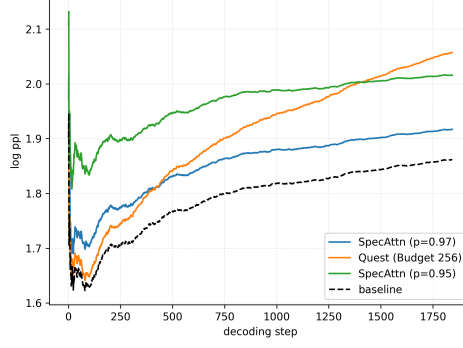
Figure 5: Perplexity (lower is better) comparison across different sparse attention methods. Here Baseline refers to the vanilla full attention decoding (StreamingLLM is omitted due to relatively high perplexity).

subject to monotonicity constraints. The monotonicity requirement ensures that $f(j_1) \leq f(j_2)$ for all $j_1 < j_2$, preserving the relative ordering of layer correspondences. Crucially, this formulation permits draft layer skipping, meaning that some draft layers may remain unmapped while others can be mapped to multiple verifier layers. This flexibility accommodates the common scenario where draft and verifier models have different depths and architectural characteristics. The problem is solved using dynamic programming where each state (i, j) represents the minimum cost of aligning draft layers up to i with verifier layers up to j, considering three transition options: diagonal matching (one-to-one layer correspondence), horizontal repetition (mapping the same draft layer to consecutive verifier layers), and vertical skipping (advancing through draft layers without correspondence). Pseudo code to solve layer mapping problem is given in Algo B.

Algorithm 3 Monotonic Dynamic Time Warping for Layer Mapping

1: **procedure** MONOTONICDTW($S \in \mathbb{R}^{m \times n}$)
2:     **Input:** Score matrix S where $S[i, j]$ is similarity between draft layer i and verifier layer j
3:     **Output:** Mapping from draft layers to verifier layers
4:     Convert to distance matrix: $D = -S$ ▷ For maximization
5:     Initialize DP table: $dp[0, 0] = 0$, all other entries $= \infty$
6:     Initialize backtrack table for path reconstruction
7:     **for** $i = 1$ to m **do**   ▷ For each draft layer
8:         **for** $j = 1$ to n **do**   ▷ For each verifier layer
9:             $cost = D[i - 1, j - 1]$
10:             **// Consider three alignment options:**
11:             $diagonal = dp[i - 1, j - 1] + cost$ ▷ Match current layers
12:             $left = dp[i, j - 1] + cost$ ▷ Repeat draft layer
13:             $above = \min_{k=0}^{i} dp[k, j - 1] + cost$ ▷ Skip draft layers
14:             $dp[i, j] = \min(diagonal, left, above)$
15:             Record best choice in backtrack table
16:         **end for**
17:     **end for**
18:     Find minimum cost in last column: $\min_i dp[i, n]$
19:     Backtrack from optimal endpoint to reconstruct path
20:     Convert path to layer mapping
21:     **return** layer mapping, total score
22: **end procedure**

C Stepwise cumulative perplexity

Fig 5 shows trends of log cumulative perplexity while decoding (on PG-19 dataset Rae et al. [2019]). Notice how Quest diverges faster than SpecAttn as the decoding step increases.