

# Scrooge: Accelerating Attention Inference in LLMs via Early Termination Mechanism

Gwangeun Byeon<sup>1</sup>, Seongwook Kim<sup>1</sup>, Taein Kim<sup>1</sup>, Jungmin Lee<sup>2</sup>, Seokin Hong<sup>1</sup>

<sup>1</sup>Department of Electrical and Computer Engineering, Sungkyunkwan University

<sup>2</sup>Department of Semiconductor and Display Engineering, Sungkyunkwan University

<sup>1,2</sup>{kebyun, su8939, xovnd2010, leejm518, seokin}@skku.edu

**Abstract**—Large Language Models (LLMs) have demonstrated remarkable performance in natural language processing and are now widely adopted in diverse applications. However, their significant computation and memory costs severely limit their acceleration. In particular, the self-attention mechanism is a significant bottleneck, as it cannot exploit batch parallelism across prompts, and its memory traffic grows quadratically with sequence length. In this paper, we propose *Scrooge*, a novel hardware accelerator framework that leverages an attention early termination mechanism, designed to address the inefficiency of self-attention. The self-attention mechanism does not assign equal importance to all tokens. Instead, semantically important tokens consistently receive higher attention scores. Consequently, preserving sufficient attention for a subset of important tokens is often enough to maintain model accuracy, even without computing attention for all tokens. Our key insight is that once sufficient attention has been accumulated, further computation with the remaining tokens only increases complexity without improving accuracy. *Scrooge* leverages this insight to approximate the attention of the remaining tokens and terminates the attention computation dynamically once it has gathered sufficient attention. With this method, *Scrooge* reduces both latency and memory traffic while maintaining accuracy. Experimental results show that *Scrooge* achieves a 1.7× speedup and a 0.47× reduction in memory traffic with negligible accuracy loss.

## I. INTRODUCTION

Transformer models have recently achieved remarkable success across diverse domains [15], [24], [30]. In particular, Large Language Models (LLMs) have exhibited significant breakthroughs in natural language processing (NLP). As the core operation of LLMs, the self-attention mechanism captures relationships between tokens within an input sequence, enabling contextual semantic representation [25]. Despite these advantages, self-attention remains a key bottleneck, as its computational complexity grows quadratically with sequence length and incurs substantial data movement [6]. Moreover, the attention mechanism offers limited batch-level parallelism since it maintains separate intermediate states (e.g., Key-Value cache) for each input sequence. These limitations restrict efficient LLM inference on conventional hardware [10], [13], [20], [29].

Several studies have been proposed to reduce the computation and memory cost of attention [5], [8], [9], [11], [16], [21], [23], [26], [31]. The attention mechanism assigns higher scores to semantically important tokens, leading to significant sparsity after the softmax operation, as most other values become zero [2], [3], [12], [18]. Leveraging this, some studies retain the KV cache only for important tokens to reduce memory

usage [23], [28], [31], while others skip computation for unimportant tokens, while others skip computation for unimportant tokens, which are near-zero values after the softmax [4], [7], [16]. Moreover, a technique that progressively prunes unimportant tokens across layers has been proposed [26], reducing the overall computational complexity. However, these approaches often face limited performance improvements due to several reasons. First, they overlook the fact that the number of important tokens varies across layers and batches. Second, many studies permanently prune tokens once they are unimportant. However, a token’s significance may change depending on the prompt, and such tokens may later become important [23]. Lastly, existing methods tend to be dependent on a specific data type, while recent LLMs utilize diverse data types [21].

In LLMs, it is not necessary to compute all tokens to maintain model accuracy. This is based on the inherent characteristic of attention mechanisms, which assign higher attention to semantically important tokens rather than allocating equal attention to all. This property allows sufficient attention to be captured from only a subset of tokens while preserving accuracy. Once sufficient attention has been accumulated, subsequent computations contribute little to accuracy but significantly increase computational complexity.

Leveraging this insight, we propose *Scrooge*, a novel hardware accelerator framework for efficient attention inference in LLMs. *Scrooge* introduces *Attention Early Termination (AET)*, which dynamically terminates the attention computation once sufficient attention has been accumulated to maintain accuracy. To this end, *Scrooge* needs to determine whether the computed attention is sufficient to preserve accuracy relative to the total attention. Instead of computing all tokens, *Scrooge* approximates the total attention by estimating the attention of remaining tokens based on the already computed tokens. The accumulated attention is then evaluated against this approximated total to determine whether it is sufficient to maintain accuracy. Once sufficient attention has been accumulated, the remaining token computations are skipped, reducing latency and memory traffic. *AET* operates independently of data types and adaptively determines the appropriate number of token computations across layers and batches to maintain accuracy. Furthermore, tokens considered unimportant in earlier prompts are not permanently excluded, ensuring that potentially relevant information is preserved.

This paper details the *Scrooge* accelerator, which implements

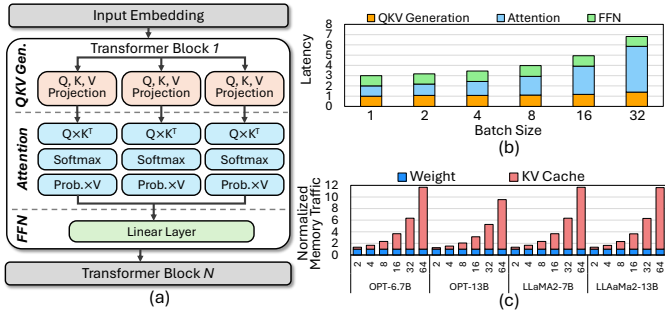


Fig. 1: (a) Basic transformer architecture. (b) Normalized latency breakdown for OPT-6.7B model. (c) Normalized off-chip memory traffic breakdown of different transformer models.

*AET*. The *Scrooge* architecture consists of a multiplier array, an adder tree, as well as an Early Termination Unit and a Value Select Unit for *AET*. To support LLMs with diverse head dimensions, *Scrooge* is designed with multi-dimensional flexibility. Depending on the model dimension, it can process up to 16 tokens in parallel. Our evaluation shows that *Scrooge* achieves an average  $1.7\times$  speedup and  $0.47\times$  off-chip memory reduction compared to conventional attention computation.

## II. BACKGROUND

### A. Large Language Model

1) *Transformer Architecture*: Large Language Model (LLM) utilizes the Transformer architecture to capture semantic meaning and contextual information. Figure 1-(a) illustrates the core mechanism of the Transformer architecture. The Transformer architecture consists of three steps: QKV Generation, Attention, and Feed-Forward Network. The following equations illustrate the detailed operations of the Transformer architecture.

$$Q, K, V = W_Q X_{in}, W_K X_{in}, W_V X_{in} \quad (1)$$

$$A_i^W = \exp\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right), A_i^S = \frac{A_i^W}{\sum_{k=1}^L A_k^W}, \text{ for } 1 \leq i \leq L \quad (2)$$

$$A^{Prob} = A^S V \quad (3)$$

The first step is QKV generation. The input embedding  $\mathbf{X}_{in} \in \mathbb{R}^{L \times D}$  is projected using the weight matrices  $W_Q$ ,  $W_K$ , and  $W_V$  to produce the query ( $\mathbf{Q}$ ), key ( $\mathbf{K}$ ), and value ( $\mathbf{V}$ ) matrices, as shown in Equation 1. The second step is Attention. First, a scaled-dot product is performed between the query matrix  $\mathbf{Q} \in \mathbb{R}^{L \times D}$  and the transposed key matrix  $\mathbf{K}^T \in \mathbb{R}^{D \times L}$  to compute the attention weight where  $A^W \in \mathbb{R}^{L \times L}$ , which is the pairwise correlation of each token. Next, each row of the attention weight is normalized using a row-wise softmax function to produce the attention score where  $A_i^S \in \mathbb{R}^{1 \times L}$  for the  $i^{\text{th}}$  row (Equation 2). Finally, the attention score is multiplied by the value matrix  $\mathbf{V} \in \mathbb{R}^{L \times D}$  to get the attention probability, where  $A^{Prob} \in \mathbb{R}^{L \times D}$  as described in Equation 3. The attention probabilities from all heads are then concatenated into a single matrix and passed to a Feed-Forward Network (FFN) layer, which produces the final output.

2) *Autoregressive Inference*: The LLM generates the token autoregressively, where the output token depends on the previous token until it generates a special  $\langle eos \rangle$  token. This process consists of a prefill and decode phase.

TABLE I: Prior Attention Optimization Techniques

	H <sub>2</sub> O [31]	SpAtten [26]	Token-Picker [21]	This Work ( <i>Scrooge</i> )
Granularity	Token-level	Token-level	Bit-level	Token-level
Layer-Awareness	No	Yes	Yes	Yes
Batch-Awareness	No	No	Yes	Yes
Permanent Pruning	Yes	Yes	No	No
Datatype Dependency	Independent	Independent	Dependent	Independent

The prefill phase processes the initial input prompt. It takes the entire prompt at once and performs a single forward pass to compute the intermediate state, known as the Key-Value (KV) cache, and produces the first new token. This phase is computationally intensive which is fully parallelized across all input tokens. In contrast, the decode phase generates a token at a time. In each decoding step, the model consumes the previously generated token and incrementally updates the KV cache. The decode phase is memory-intensive and has limited parallelism, which often makes it the dominant factor in inference latency for long sequence generation.

### B. Challenges of Attention Computation

In the decode phase, the LLM requires QKV and FFN weights as well as the KV cache for attention from off-chip memory. The sizes of these weights increase with the number of layers, heads, and dimensions, while the KV cache size grows with the input sequence length, resulting in substantial memory traffic and making memory access a major bottleneck. To mitigate this overhead, batching is commonly used. Using batch processing, QKV generation and FFN share weights across different batches. This allows multiple inputs to be computed together, amortizing memory access for weights and enabling parallel processing. In contrast, attention requires a unique KV cache per batch, making batch-level parallelism infeasible. As a result, both the latency and memory traffic of the attention operation increase with batch size and sequence length, which significantly restricts acceleration.

Figure 1-(b) shows the latency breakdown of the OPT-6.7B model with 1024 sequence length across different batch sizes. While the latency for QKV generation and FFN does not increase significantly due to batch-level parallelism, attention latency increases with batch size because each batch requires a separate KV cache. Figure 1-(c) further shows the memory traffic of weights and the KV cache with the batch size. Unlike weights, the KV cache is not shared across batches, leading to a quadratic increase in memory traffic. Therefore, optimization techniques are necessary to address the attention bottlenecks.

## III. MOTIVATION

### A. Limitations of Prior Works

Several software and hardware based attention optimization techniques have been proposed, as summarized in Table I. H<sub>2</sub>O [31] proposed a KV cache compression technique to reduce complexity and memory traffic. It retains KV cache for tokens based on historical attention scores within a fixed KV cache budget to preserve important context while reducing memory usage. However, H<sub>2</sub>O overlooks that the required number of tokens varies by batch and layer depth, and permanently evicts tokens even if they are later needed [23].

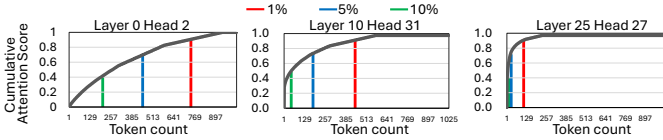


Fig. 2: Cumulative attention score (after softmax) for OPT-6.7B model. We select one head from the early, middle, and late layers. The vertical lines indicate the cumulative attention scores at which accuracy decreases by 1% (red), 5% (blue), and 10% (green), respectively.

SpAtten [26] proposed a cascading token pruning method that progressively prunes tokens at each layer based on a predefined ratio. However, SpAtten does not consider that the number of important tokens can vary across different batches. In addition, pruned tokens are permanently removed and cannot contribute even if they become important later.

Token-Picker [21] skips the computation of unimportant tokens through partial-bit computation, considering token importance at both the layer and batch levels. However, it operates only on signed integers and is not easily applicable to other data types. Furthermore, fetching partial bits leads to non-coalesced memory accesses, which degrade efficiency.

### B. Diminishing Returns of Full Attention

In Transformer models, the attention mechanism does not assign equal attention scores to all tokens. Instead, it allocates higher scores to tokens that are more semantically important within the context. Preserving sufficient attention for these important tokens is often enough to maintain model accuracy.

Figure 2 presents the cumulative attention score for the OPT-6.7B model with an input length of 1,024 tokens. The cumulative attention score across all tokens sums to 1 (Equation 2). The vertical lines indicate the cumulative attention score at which the accuracy loss is 1%, 5%, and 10%. For example, the green line for layer 0 head 2 shows that the attention scores for 194 of the 1,024 tokens account for approximately 40% of the total attention score, with an accuracy loss of less than 10%.

As shown in the figure, computing only 731, 417, and 103 tokens out of 1,000 for layers 0, 10, and 25, respectively, preserves model accuracy with a loss of less than 1%. Moreover, the number of tokens required to maintain accuracy decreases in deeper layers. This is because, while tokens in earlier layers have similar attention scores, in deeper layers, a small subset of tokens dominates the total attention. Consequently, this indicates that once sufficient attention has been accumulated, computing the remaining tokens increases computational complexity without improving accuracy, particularly in later layers.

## IV. SCROOGE: ATTENTION EARLY TERMINATION

The main goal of the *Scrooge* framework is to reduce the latency and memory traffic of attention operations through *attention early termination*. As discussed in Section III-B, attention from a subset of tokens is often enough to maintain accuracy without processing all tokens. Thus, once sufficient attention has been accumulated, the remaining token computations can be skipped, reducing computational complexity. To this end, *Scrooge* determines whether the attention accumulated from computed tokens is sufficient compared to the total

attention. The total attention, which corresponds to the softmax denominator, requires the attention weights of all tokens, both computed and uncomputed. Rather than computing all tokens, *Scrooge* approximates the attention of the uncomputed tokens based on the already computed ones to estimate the total attention. The attention computation is terminated early if the ratio of the accumulated attention to the total approximated attention exceeds a predefined threshold.

The following sections provide a detailed description of this process. The first is the computation of important tokens to quickly accumulate attention (Section IV-A). The second is the approximation of the expected attention of the remaining tokens to determine whether to terminate early (Section IV-B).

### A. Prioritizing Token Computation

In LLMs, semantically important tokens tend to have higher attention scores. By computing these tokens first, sufficient attention can be accumulated with fewer token computations. We categorize important tokens into three types. The first is the initial token, or sink token, which often exhibits strong attention despite low semantic importance [28]. The second is the global token, a historically high-attention token [31], based on the intuition that tokens important in the current query often remain important in subsequent queries. The third is the recent token, referring to the most recently generated token. Due to the sequential nature of generation, recent tokens strongly influence the attention distribution of subsequent tokens.

The goal of *Scrooge* is to accumulate as much attention as possible with the minimal number of token computations. To this end, *Scrooge* prioritizes computing the sink token, global tokens, and recent tokens first. Unlike prior techniques [31] that store as many tokens as the cache budget allows, *Scrooge* retains only 64 global tokens and 8 recent tokens. Computing important tokens first allows *Scrooge* to accumulate sufficient attention with minimal computations. Their average attention weight is then leveraged to approximate the remaining and total attention, which is used to determine the early termination condition, as will be discussed in the following sections.

### B. Attention Early-Termination Mechanism

The main idea of *Attention Early Termination (AET)* is to determine whether the accumulated attention is sufficient to preserve accuracy compared to the total attention. However, computing the total attention, which corresponds to the softmax denominator, requires processing all tokens. To address this, we approximate the total attention by estimating the attention of the remaining uncomputed tokens (unknown tokens) based on the attention of the already computed ones (known tokens). *Scrooge* first prioritizes computing important tokens, as they typically have larger attention weights. Consequently, the attention of the unknown tokens can be conservatively assumed to be less than or equal to that of the important tokens. By leveraging the average attention weight of the known tokens, *Scrooge* conservatively approximates the attention weight of the unknown tokens.

Figure 3-(a) illustrates an example of approximating the attention weights of three unknown tokens. The approximation

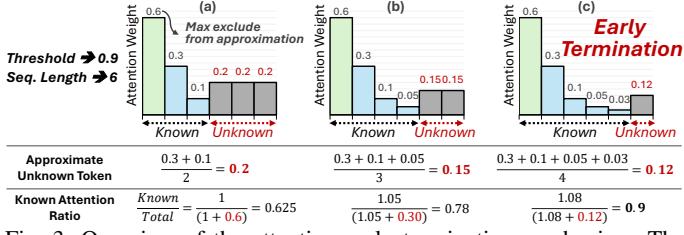


Fig. 3: Overview of the attention early termination mechanism. The known token attention weight is used to approximate the attention weight of unknown tokens. In this example, the threshold is set to 0.9 and the sequence length is 6.

is performed by averaging the attention weights of two known tokens while excluding the maximum value among the three known tokens. In LLMs, one or two semantically important tokens often account for 80–90% of the total attention score after the softmax operation. As a result, the remaining tokens typically exhibit small attention scores with a significant difference from the maximum. Including the maximum value in the approximation can therefore introduce significant bias, whereas excluding it provides a conservative and less biased estimate of the unknown attention. In Figure 3-(a), the unknown token attention weights are approximated as 0.2 by averaging 0.3 and 0.1 while excluding the maximum token value of 0.6.

The early termination condition is determined by whether the ratio of the accumulated attention from known tokens to the total expected attention exceeds a predefined threshold. In Figure 3-(a), the sum of three known token attentions is 1.0 (0.6+0.3+0.1) and the sum of three approximated unknown token attentions is 0.6 (0.2+0.2+0.2), resulting in a total approximated attention of 1.6. The ratio of the known token attention weight is 0.625 ( $= 1/1.6$ ), which falls below the threshold of 0.9 and requires additional token computations. In contrast, in Figure 3-(c), the ratio of known token attention weight reaches 0.9, which equals the threshold, allowing for an early termination that skips the computation of one remaining token.

### C. Detailed Scrooge Algorithm

Algorithm 1 illustrates the detailed AET algorithm. The first step is the computation of important tokens (lines 1–5). The sink, local, and global token indices are added to  $S_{\text{known}}$  (line 1), and their attention weights are computed (line 4). The next step is the approximate unknown attention and early termination. *Scrooge* computes tokens in reverse order, starting from the most recent token to the oldest one (line 6). For each iteration, the sum and average values of the attention weights for the known tokens are updated (lines 11–12). Using this average value, the attention of the unknown tokens is approximated, and the ratio of known attention to total expected attention is evaluated against the threshold (lines 13–14). If the ratio exceeds the threshold, the attention computation is terminated early (line 15). After early termination, a replacement step is performed between the global buffer and the local token buffer. In this step, the token with the smallest attention weight in the global buffer is removed, and the oldest token from the local buffer is promoted to the global buffer (lines 18–19). Finally, a

### Algorithm 1 Scrooge: Attention Early Termination Algorithm

```

Input:  $Q, K, V, L, thr_k, thr_v$   $\triangleright$  Query, Key, Value, Sequence length, Threshold
Input:  $\mathbb{L}, \mathbb{G}$   $\triangleright$  Local and Global token indices
Output:  $A^{\text{Score}}, S_{\text{value}}$ 

/* Step 1: Important Token Computation */
1:  $S_{\text{known}} \leftarrow \text{Sink} \cup \mathbb{L} \cup \mathbb{G}$ 
2:  $S_{\text{value}} \leftarrow \text{Sink} \cup \mathbb{L} \cup \mathbb{G}$   $\triangleright$  For the value tensor optimization
3:  $n = \text{Length}(S_{\text{known}})$ 
4:  $A_x^W = \text{EXP}(Q_x K_x^T), \forall x \in S_{\text{known}}$ 
5:  $\text{Max} = \text{MAX}(A_{S_{\text{known}}}^W)$   $\triangleright$  Maximum value is the bias

/* Step 2: Attention with Early Termination */
6: for  $i = L - 1$  down to 0 where  $i \notin S_{\text{known}}$  do
7:    $S_{\text{known}} \leftarrow i$ 
8:    $n += 1$ 
9:    $A_i^W \leftarrow \text{EXP}(Q_i K_i^T), A_{S_{\text{known}}}^W \leftarrow A_i^W$ 
10:   $S_{\text{value}} \leftarrow i$  if  $A_i^W \geq \text{Max} \times thr_v$   $\triangleright$  For value optimization
11:   $\text{Sum}_K = \text{SUM}(A_{S_{\text{known}}}^W)$   $\triangleright$  Sum of known attention
12:   $\text{Avg} = (\text{Sum} - \text{Max}) / (n - 1)$   $\triangleright$  Approximate unknown token
13:   $\text{Sum}_U = \text{Avg} \times (L - n)$   $\triangleright$  Sum of unknown attention
14:  if  $(\text{Sum}_K + \text{Sum}_U) \times thr_k \leq \text{Sum}_K$  then  $\triangleright$  Known attention ratio
15:    break: Early Termination
16:  end if
17: end for

/* Step 3: Important token replacement */
18:  $i \leftarrow \text{minIndex}(\mathbb{L})$   $\triangleright$  Pop the token with smallest index
19:  $j \leftarrow \text{argmin}_{x \in \mathbb{G}} (\sum A_x^W)$   $\triangleright$  Pop the smallest accumulated value token
20:  $\mathbb{L}.\text{Remove}(i), \mathbb{G}.\text{Remove}(j), \mathbb{G}.\text{Insert}(i)$   $\triangleright$  Replacement

/* Step 4: Softmax */
21:  $A^{\text{Score}} = \text{Softmax}(A_{S_{\text{known}}}^W)$ 
22: Return  $A^{\text{Score}}, S_{\text{value}}$ 

```

softmax operation is applied to the attention weight of known tokens to produce the final attention scores (lines 21–22).

*Scrooge* also optimizes  $\mathbf{V}$  tensor computation by leveraging the characteristic of the softmax function. Since softmax amplifies relative differences, tokens with attention weights much smaller than the maximum result in near-zero values and have a negligible impact on the output. To leverage this property, *Scrooge* skips tokens with attention weights below  $\text{Max} \times thr_v$ , where  $\text{Max}$  is determined at line 5. Tokens exceeding this threshold are selected, and their indices are stored in  $S_{\text{value}}$  (line 10). The  $\mathbf{V}$  tensor computation is then performed only on these tokens, thereby reducing the  $\mathbf{V}$  tensor computation.

## V. SCROOGE ARCHITECTURE

### A. Overview

Figure 4 illustrates the overview of *Scrooge* architecture, which implements an AET algorithm using the Early Termination Unit (ETU) and the Value Select Unit (VSU). The *Scrooge* architecture can process multiple tokens in parallel based on the head dimension. For example, with a head dimension of 128 (e.g., OPT-6.7B, LLaMA-2-7B), it can handle up to eight tokens simultaneously.

For the attention computation, the  $\mathbf{Q}$  tensor is loaded into the Q Buffer, while the  $\mathbf{K}$  tensors for the sink, local, and global tokens are loaded into the KV Buffer, Local Buffer, and the Global Buffer (①), respectively. The fetched  $\mathbf{Q}$  tensors and  $\mathbf{K}$  tensors of important tokens are multiplied in the Multiplier

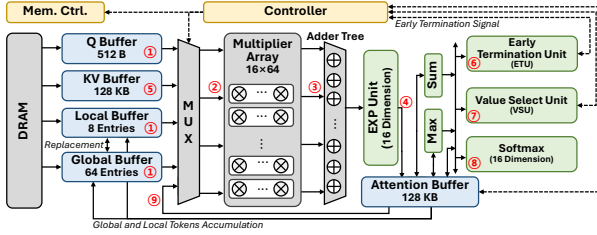


Fig. 4: *Scrooge* architecture overview.

Array (②). When the head dimension is larger than the computation dimension, multiple computation lanes can be allocated to a single token operation. The Adder Tree aggregates the multiplication results for each token operation and produces the final attention weight (③). Each attention weight is applied to an exponent operation and then stored in the Attention Buffer (④). The exponential operation is approximated using a 5<sup>th</sup> order Taylor expansion [19]. The Sum and Max Unit computes the sum and the maximum of attention weights required to evaluate the early termination condition.

After the computation of important tokens, the **K** tensors of the remaining unknown tokens are fetched into the KV Buffer and their attention weights are subsequently computed (⑤). The ETU evaluates early termination conditions by approximating the expected total attention with the average computed from the Max and Sum values (⑥). Concurrently, the VSU determines whether the **V** tensor computation of a token should be executed by comparing with the maximum value and a predefined threshold, and stores the corresponding token index in the Attention Buffer (⑦). After the AET, the softmax operation is applied to the tokens in the Attention Buffer to compute their attention scores (⑧). These scores are then multiplied by the **V** tensors of the tokens fetched into the KV Buffer to produce the final attention probabilities (⑨).

### B. Flexible Design of *Scrooge* Architecture

The key design consideration of *Scrooge* architecture is dimensional flexibility. The head dimensions of major LLMs vary from 64 in OPT-1.3B to 128 in LLaMA2-7B. To support inference across models with different head dimensions, *Scrooge* employs sixteen 64-dimensional computation lanes, allowing for the parallel processing of up to sixteen tokens. Furthermore, the major components use a tree-based computation unit to enable parallel operations and flexibly support the AET mechanism.

Figure 5 illustrates an example of a detailed architecture that supports parallel token processing. The Multiplier Array consists of four two-dimensional computation lanes, enabling it to

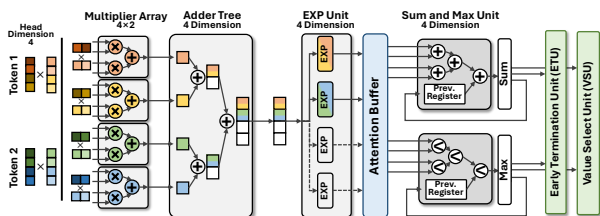


Fig. 5: Example execution flow of the core computation in the *Scrooge* architecture. *Scrooge* architecture in this example can process four tokens simultaneously and demonstrates the parallel computation of two tokens with a head dimension of 4.

TABLE II: *Scrooge* Architecture Configuration

PE Lane	16 64-dim multiplier array, 16-dim adder tree
Exp Unit	16 dimension exponent unit
Sum and Max Unit	16 dimension adder & comparator tree
On-chip Buffer	512B for query buffer, 128KB for KV and attention buffer, 8 and 64 entries for global and local buffer
Main Memory	HBM2, 128 GB/s, 16×64-bits HBM channel

process up to four tokens simultaneously. The figure shows an example of computing two tokens, both with a head dimension of four. The dimension of each token is split into two chunks to match the 2-dimensional Multiplier Array. Each chunk passes through the Multiplier Array and is then aggregated in the Adder Tree with the chunks of the same token. The computation of the Max and Sum values, which are critical parameters for estimating the AET condition, is implemented using a tree-based architecture that enables the parallel processing of multiple tokens. When Max and Sum operations are restricted to a single token, their lower throughput compared to the multiplier array would cause bottlenecks, degrading overall performance. As a result, the *Scrooge* architecture provides high flexibility across varying head dimensions in LLMs.

## VI. EVALUATIONS

### A. Methodology

1) *Algorithm Implementation*: To evaluate *Scrooge*, we conduct experiments on OPT models ranging from 1.3B to 13B parameters and LLaMA-2 models with 7B and 13B parameters. Model accuracy is evaluated using perplexity (PPL) on the Wikitext-2 dataset [17], where lower values indicate better performance. We implement the *Scrooge* algorithm inside the model’s *forward* function in the Huggingface framework [27] to evaluate algorithm performance.

2) *Hardware Implementation*: We implement the *Scrooge* architecture in SystemVerilog based on the configuration in Table II and synthesize it using Synopsys Design Compiler with a UMC 28nm standard cell library, targeting 1 GHz. The SRAM buffer is modeled using CACTI 7.0 [1] with two banks and a single read/write port per bank. Performance and memory traffic are evaluated using a custom cycle-accurate simulator built on SST [22] and integrated with DRAMSim3 [14]. *Scrooge* uses FP16, a widely adopted data type for LLMs.

3) *Comparison*: For comparison, we use a baseline design without Global and Local tokens, the Early Termination Unit, and the Value Select Unit. *Scrooge* is evaluated in two configurations: *Scrooge* (S), which maintains perplexity within  $\pm 0.5\%$ , and *Aggressive Scrooge* (AS), which allows up to a 5% perplexity increase for speedup. For the OPT models, *Scrooge* uses  $\text{threshold}_K/\text{threshold}_V$  of 0.95/0.001 for the

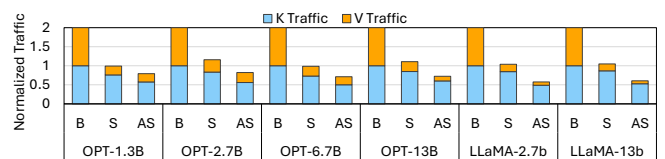


Fig. 6: Off-chip memory traffic comparison between Baseline (B), *Scrooge* (S), and *Aggressive Scrooge* (AS).

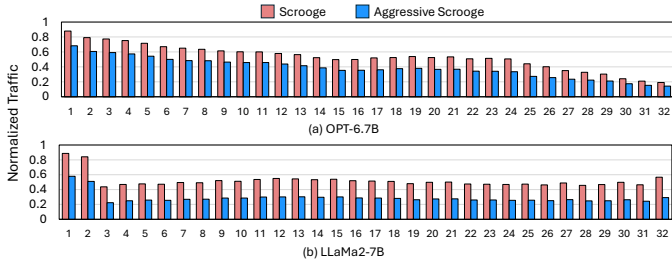


Fig. 7: Off-chip memory traffic reduction across layers for (a) OPT-6.7B and (b) LLaMa2-7B models.

1.3B and 2.7B models, and 0.90/0.001 for the 6.7B and 13B models, while *Aggressive Scrooge* applies 0.80/0.001 (1.3B), 0.75/0.005 (2.7B), 0.70/0.001 (6.7B), and 0.70/0.005 (13B). For the LLaMA-2 models, *Scrooge* applies 0.90/0.001 for both 7B and 13B, and *Aggressive Scrooge* uses 0.60/0.005 for both configurations. All experiments use autoregressive decoding, starting from a sequence length of 1 and generating up to 1024.

## B. Experimental Results

1) *Memory Traffic and Speedup*: Figure 6 shows the off-chip memory traffic of *Scrooge*. On average, *Scrooge* (S) and *Aggressive Scrooge* (AS) reduce off-chip traffic by 1.92 $\times$  and 2.94 $\times$ , respectively, compared to the baseline. Specifically,  $K$  tensor traffic is reduced by 1.23 $\times$  (S) and 1.85 $\times$  (AS), while  $V$  tensor traffic is reduced by 4.34 $\times$  (S) and 7.14 $\times$  (AS). The  $K$  tensor typically exhibits a lower off-chip memory reduction than the  $V$  tensor. This is because the  $K$  tensor contributes to the softmax denominator, which is shared across all tokens and thus limits optimization opportunities. In contrast, the  $V$  tensor contributes independently to each token output, allowing more aggressive optimization.

Figure 7 illustrates the off-chip memory traffic for each layer of the OPT-6.7B and LLaMa-2-7B models. Memory traffic significantly decreases in later layers compared to earlier layers in both models. This is because, in the earlier layers, the similar attention scores across tokens result in comparable contributions to the output, thereby limiting opportunities for optimization. In contrast, in the later layers, a small subset of tokens accounts for 80–90% of the total attention score, enabling sufficient attention to be accumulated with fewer token computations. This result underscores that layer-aware optimization is required since the opportunity to skip token computations differs across layers.

2) *Speedup*: Figure 8 shows the speedup of the *Scrooge* architecture across different models. *Scrooge* achieves a 1.7 $\times$  speedup over the baseline, while *Aggressive Scrooge* achieves a 2.6 $\times$  speedup. The main reason for this improvement is the significant reduction in  $K$  and  $V$  tensor computations enabled by *AET*. Furthermore, *Aggressive Scrooge* achieves an additional 1.4 $\times$  speedup over *Scrooge*, trading off up to a 5% increase in perplexity.

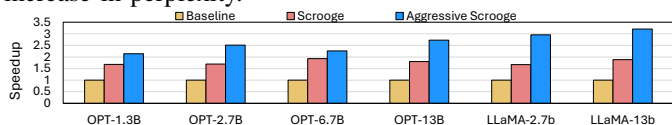


Fig. 8: Performance comparison between baseline, *Scrooge*, and *Aggressive Scrooge*.

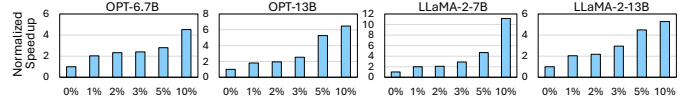


Fig. 9: Speedup of *Scrooge* under different levels of perplexity loss. The x-axis indicates that the perplexity loss is equal to or less than the corresponding percentage.

Figure 9 illustrates the trade-off between perplexity loss and speedup for the OPT-6.7B/13B and LLaMA-2-7B/13B models. On average, our technique achieves a 1.97 $\times$  speedup with less than a 1% perplexity loss. The speedup increases to 2.13 $\times$ , 2.70 $\times$ , 4.31 $\times$ , and 6.86 $\times$  for perplexity losses of 2%, 3%, 5%, and 10%, respectively. These results demonstrate that our technique can effectively improve performance by exploiting the trade-off between perplexity and speedup.

TABLE III: Hardware Overhead

Component	Area ( $mm^2$ )	Power (mW)
Multiplier Array	1.138 (15.10%)	1405.20 (55.95%)
Adder Tree	0.024 (0.32%)	21.36 (0.85%)
EXP Unit	0.116 (1.54%)	79.78 (3.18%)
Max & Sum Unit	0.009 (0.13%)	10.98 (0.44%)
Softmax	2.176 (28.86%)	134.79 (5.37%)
ETU & VSU	0.005 (0.07%)	3.67 (0.14%)
SRAM Buffer	4.071 (53.99%)	855.8 (34.07%)
Total	7.539	2511.59

3) *Hardware Overhead*: As shown in Table III, the total chip area of *Scrooge* is  $7.54mm^2$ . The SRAM buffer occupies the largest portion, accounting for 53.99% of the total area. The ETU and VSU, which are additional hardware components for *Scrooge*, account for only 0.07% of the total area. This low overhead is due to their simple component design: the ETU consists of two adders, two multipliers, one comparator, and one divider, while the VSU contains one adder, one comparator, and one multiplier. While the tree structure demanded for dimensional flexibility incurs a particular area and power overhead, this is effectively offset by the significant reduction in off-chip memory access for  $K$  and  $V$  vectors.

## VII. CONCLUSION

In this paper, we addressed the attention latency and memory traffic problem in LLM inference. We observed that computing all tokens is not necessary to preserve model accuracy. Leveraging this insight, we introduced *Scrooge*, a novel attention accelerator framework. *Scrooge* efficiently optimizes attention latency and memory traffic by computing only enough attention to preserve perplexity. Our evaluations show that *Scrooge* achieves a 1.7 $\times$  speedup and a 0.47 $\times$  reduction in off-chip memory compared to conventional attention computation.

## ACKNOWLEDGMENT

This work was partly supported by the Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.10692981, 50% / No.00228970, 25%) and the IITP (Institute of Information & Communications Technology Planning & Evaluation)-ITRC (Information Technology Research Center) (RS-2021-II212052, 25%) grant funded by the Korea government (Ministry of Science and ICT). Seokin Hong is the corresponding author.

## REFERENCES

- [1] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.
- [2] G. Byeon, S. Kim, H. Kim, S. Han, J. Kim, P. Nair, T. Kang, and S. Hong, "Avalanche: Optimizing cache utilization via matrix reordering for sparse matrix multiplication accelerator," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 1746–1759.
- [3] G. Byeon, S. Lee, S. Kim, Y. Kim, P. J. Nair, and S. Hong, "Sparseft: Sparsity-aware fault tolerance for reliable cnn inference on gpus," in *2023 32nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2023, pp. 337–338.
- [4] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.
- [5] H. Cho, Y. Jang, H. Kim, S. Kim, K. Kwon, G. Kim, and S. Hong, "Librapim: Dynamic load rebalancing to maximize utilization in pim-assisted llm inference systems," in *2025 34th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2025, pp. 43–56.
- [6] K. Choromanski, V. Likhoshesterov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser *et al.*, "Rethinking attention with performers," *arXiv preprint arXiv:2009.14794*, 2020.
- [7] T. J. Ham, S. J. Jung, S. Kim, Y. H. Oh, Y. Park, Y. Song, J.-H. Park, S. Lee, K. Park, J. W. Lee *et al.*, "A<sup>3</sup>: Accelerating attention mechanisms in neural networks with approximation," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 328–341.
- [8] T. J. Ham, Y. Lee, S. H. Seo, S. Kim, H. Choi, S. J. Jung, and J. W. Lee, "Elsa: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 692–705.
- [9] S. Han, S. Kim, G. Byeon, J. Yoon, and S. Hong, "Zebra: Leveraging diagonal attention pattern for vision transformer accelerator," in *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 2025, pp. 1–7.
- [10] S. Jang, J. Park, Y. Lee, O. Kwon, D. Kim, J. Seok, and S. Hong, "Softwalker: Supporting software page table walk for irregular gpu applications," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, 2025, pp. 401–417.
- [11] Y. Jang, H. Cho, Y. Ryu, J. Kim, and S. Hong, "Pimpal: Accelerating llm inference on edge devices via in-dram arithmetic lookup," in *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2025, pp. 1–7.
- [12] S. Kim, G. Byeon, S. Kim, H. Kim, and S. Hong, "Conveyor: Towards asynchronous dataflow in systolic array to exploit unstructured sparsity," in *2023 IEEE 41st International Conference on Computer Design (ICCD)*. IEEE, 2023, pp. 423–431.
- [13] O. Kwon, Y. Lee, J. Park, S. Jang, B. Tak, and S. Hong, "Distributed page table: Harnessing physical memory as an unbounded hashed page table," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 36–49.
- [14] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "Dramsim3: A cycle-accurate, thermal-capable dram simulator," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 106–109, 2020.
- [15] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10012–10022.
- [16] L. Lu, Y. Jin, H. Bi, Z. Luo, P. Li, T. Wang, and Y. Liang, "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 977–991.
- [17] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," 2016.
- [18] A. Navarro, J. Cano, J. L. Abellan, and M. E. Acacio, "Qufi: Adaptive tiled gustavson output reuse for edge sparse dnn accelerators," in *2025 IEEE 43rd International Conference on Computer Design (ICCD)*, 2025, pp. 119–126.
- [19] P. Nilsson, A. U. R. Shaik, R. Gangarajiah, and E. Hertz, "Hardware implementation of the exponential function using taylor series," in *2014 NORCHIP*. IEEE, 2014, pp. 1–4.
- [20] J. Park, O. Kwon, Y. Lee, S. Kim, G. Byeon, J. Yoon, P. J. Nair, and S. Hong, "A case for speculative address translation with rapid validation for gpus," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 278–292.
- [21] J. Park, M. Kang, Y. Han, Y.-G. Kim, J. Shin, and L.-S. Kim, "Token-picker: Accelerating attention in text generation with minimized memory transfer via probability estimation," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.
- [22] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. Cooper-Balis *et al.*, "The structural simulation toolkit," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 4, pp. 37–42, 2011.
- [23] J. Tang, Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, and S. Han, "Quest: Query-aware sparsity for efficient long-context llm inference," *arXiv preprint arXiv:2406.10774*, 2024.
- [24] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.
- [25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
- [26] H. Wang, Z. Zhang, and S. Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 97–110.
- [27] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.
- [28] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," *arXiv preprint arXiv:2309.17453*, 2023.
- [29] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for {Transformer-Based} generative models," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 521–538.
- [30] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.
- [31] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett *et al.*, "H2o: Heavy-hitter oracle for efficient generative inference of large language models," *Advances in Neural Information Processing Systems*, vol. 36, pp. 34 661–34 710, 2023.