SubKV: Quantizing Long Context KV Cache for Sub-billion Parameter Language Models on Edge Devices

Ziqian Zeng[1] | Tao Zhang[1] | Zhengdong Lu[1] | Wenjun Li[1] | Huiping Zhuang[1] | Hongen Shao[2] | Sin G. Teo[3] | Xiaofeng Zou[2]

[1]Shien-Ming Wu School of Intelligent Engineering, South China University of Technology, Guangdong, China

[2]School of Future Technology, South China University of Technology, Guangdong, China

[3]Institute for Infocomm Research, A*Star, Singapore

Correspondence
Xiaofeng Zou, School of Future Technology, South China University of Technology, China.
Huiping Zhuang, Shien-Ming Wu School of Intelligent Engineering, South China University of Technology, China.
Email: zouxiaofeng@scut.edu.cn, hpzhuang@scut.edu.cn

Abstract

Large Language Models (LLMs) have demonstrated remarkable capabilities across various tasks. However, their substantial computational and memory requirements present significant challenges for widespread deployment on edge devices. In long context scenarios, even sub-billion parameter LLMs encounter unavoidable memory and performance bottlenecks. This paper addresses these challenges by introducing advanced quantization techniques tailored for sub-billion parameter LLMs. It specifically targets reducing memory consumption through the conversion of the model's KV Cache to lower-bit integers. We present SubKV, a quantization method specifically designed to optimize the KV Cache in sub-billion parameter LLMs. Our analysis reveals distinct distributional differences in the magnitude of key and value caches. Leveraging this insight, we apply Per-Channel Quantization to the key cache and Per-Token Quantization to the value cache. Furthermore, we introduce the Dynamic Window Quantization method to enhance attention computations. To mitigate the extreme sensitivity of the first token, we also introduce Attention Sink-Aware Quantization. Experimental results demonstrate that SubKV significantly reduces the KV Cache size during long context inference while maintaining model performance, offering superior results to existing KV Cache quantization methods.

KEYWORDS
KV Cache Quantization, Per-Channel Key Quantization, Per-Token Value Quantization, Dynamic Window Quantization, Sink-Aware Quantization, Edge Devices.

1 | INTRODUCTION

Large Language Models (LLMs)[1,2,3,4,5], comprising billions or even trillions of parameters, are trained on vast corpora of textual data. LLMs consistently exhibit remarkable performance across various tasks, but their exceptional capabilities come with significant challenges stemming from their extensive size and computational requirements. For instance, the GPT-175B model[2], with an impressive 175 billion parameters, demands a minimum of 350 GB of memory in half-precision (FP16) format. Furthermore, deploying this model for inference necessitates at least five A100 GPUs 80GB to efficiently manage operations. Even comparatively smaller models, such as LLaMA-2-7B[2], which use 8-bit quantization, remain prohibitively resource-intensive for edge devices. Moreover, considerations of portability and computational cost underscore the urgent need to enable LLM deployment on edge devices.

Abbreviations: LLM, Large Language Model;NLP, Natural Language Processing; MLM, Masked Language Model; GPT, Generative Pre-trained Transformer; BERT, Bidirectional Encoder Representations from Transformers; RNN, Recurrent Neural Network; LSTM, Long Short-Term Memory; CNN, Convolutional Neural Network; VAE, Variational Autoencoder; GAN, Generative Adversarial Network; Attention, Attention Mechanism; Transformer, Transformer Model; KV Cache, Key-Value Cache; Seq2Seq, Sequence to Sequence; Self-Attention, Self-Attention Mechanism; RoBERTa, A Robustly Optimized BERT Pretraining Approach; T5, Text-To-Text Transfer Transformer; RLHF, Reinforcement Learning from Human Feedback; MoE, Mixture of Experts; SOTA, State of the Art; FP16, 16-bit Floating Point Precision; FP32, 32-bit Floating Point Precision; INT8, 8-bit/4-bit Integer Precision; INT4, 4-bit Integer Precision; Distillation, Knowledge Distillation; Zero-shot, Zero-shot Learning; Few-shot, Few-shot Learning; Prompting, Prompt Engineering; Embeddings, Word/Token Embeddings.

To tackle these issues, a direct solution involves designing and implementing LLMs with parameters under one billion. Fortunately, several highly efficient sub-billion parameter models have emerged, including MobileLLM[6], H2O-Danube3[7]. These models offer practical alternatives for on-device inference, enabling real-time processing on modern smartphones. With advancements in LLMs, the demand for extended context length has grown, empowering them to address sophisticated tasks such as legal contract analysis[8] and comprehensive reviews of the literature[9]. LLM inference operates in an auto-regressive manner, generating sentences token by token. To reduce the computation overhead, the inference system always stores the key and value activations in memory and reuses them during subsequent token generation steps. The stored data is known as key and value cache (KV Cache). With the increasing popularity of utilizing LLM for long context tasks, the KV Cache consumes a significant amount of memory. On the other hand, the large amount of KV Cache can also bring a large amount of memory access in the attention mechanism when generating the output tokens. The system will be stuck on the memory access, known as the memory-bound problem in LLM inference[10]. Recent advances in IoT optimization[11], edge computing[12], and cloud-edge collaboration[13,14,15] highlight the importance of resource-aware design, yet memory-bound LLM inference at the edge devices remains underexplored.

To reduce the size of the KV Cache, a straightforward and effective approach is to reduce the total bytes consumed by the cache through quantization[16]. Unlike the extensively studied weight quantization[17,18], there are only a limited number of studies that have applied quantization to the KV Cache[19,20,21], largely due to the streaming nature of KV Cache and other complexities. Furthermore, existing research on KV Cache primarily focuses on multi-billion scale models, resulting in a notable gap in studies exploring KV Cache quantization within sub-billion parameter LLMs.

In this paper, we observe that there is a significant difference in the distribution of key cache and value cache in sub-billion parameter LLMs. This corroborates previous observations about outlier channels in LLM activations[22,18,19,20]. The difference in the distribution of key cache and value cache has a great impact on quantization accuracy, especially in extremely low-bit width scenarios. Based on this observation, we propose **SubKV**, a plug-and-play extreme low-bit KV Cache quantization method. SubKV quantizes key cache per-channel and quantizes value cache per-token.

Due to the locality of attention, recently generated KV Cache is likely to be attended to with high probability[23,24,25]. We propose the Dynamic Window Quantization strategy that preserves a small portion of the most recently generated KV Cache from quantization. During inference, as each new token is generated, the attention weights assigned to the KV Cache of earlier tokens decrease substantially. This allows the quantization of these tokens with minimal impact on model accuracy. We determine whether to quantize the edge tokens of the window or extend the window length by comparing the attention scores of newly generated tokens with those at the window's edge. This approach is efficient and straightforward to implement within existing inference systems, making it practical for real-world deployment.

Maintaining a small number of sink tokens at high precision has been proven to be an effective strategy. This approach is motivated by previous studies showing that significant attention scores are disproportionately allocated to the first few tokens after the initial layers[26]. As the positions of these sink tokens are fixed, this approach is straightforward to implement, and we incorporate it into our experiments. In this work, we represent the first token in FP16 format, achieving notable reductions in perplexity, particularly in 2-bit quantization.

To evaluate the effectiveness of our method, we conducted experiments on models of H2O-Danube3-500M[7], MobileLLM-600M[6], MobileLLM-1B[6] and Tinyllama-1.1B-chat[27]. The experiment results have showed that our proposed methods can quantize the key and value cache into 4 bits with almost no perplexity degradation on both WikiText2[28] and C4[29]. Furthermore, on the WikiText2 benchmark, our method achieves only 0.31 perplexity degradation under 2-bit quantization, significantly outperforming KVQuant (1.99)[20], KIVI (0.54)[19], and SKVQ (0.51)[30]. This demonstrates a 39.2% relative improvement over the nearest competitor while maintaining similar compression rates. Our contributions are summarized as follows:

- We observe that there exist significant differences in the outlier patterns of KV Cache in sub-billion parameter LLMs. This insight suggests that the key cache in sub-billion parameter LLMs should be quantized per-channel, while the value cache should be quantized per-token.
- We propose SubKV, a novel plug-and-play extreme low-bit KV Cache quantization method that differentiates quantization strategies for key and value cache. We introduce a Dynamic Window Quantization strategy that optimally preserves recently generated KV Cache. Additionally, based on the importance of the first few tokens, we incorporate Attention Sink-Aware Quantization to enhance performance.
- We empirically demonstrate that our proposed method can quantize the key and value cache into 4 bits with almost no perplexity degradation and achieve only 0.31 perplexity degradation with 2-bit quantization, outperforming previous quantization methods across various average bit widths.

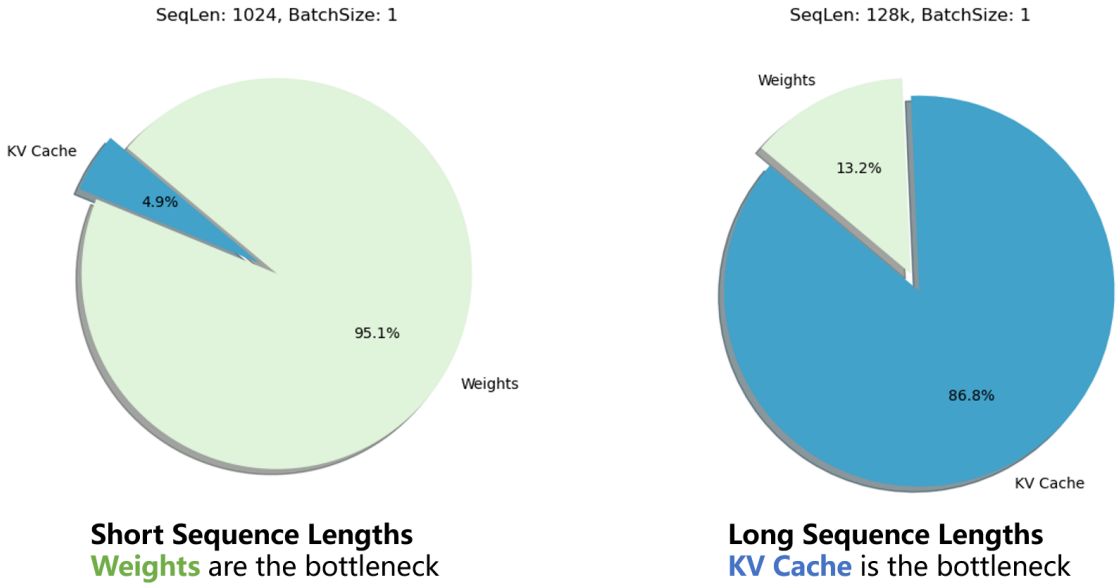

FIGURE 1 Model size and activation memory size for different sequence lengths. For long sequence lengths and large batch sizes, activation memory is the main bottleneck. At a sequence length of 128K with the H2O-Danube3-500M model, the KV Cache is the main bottleneck.

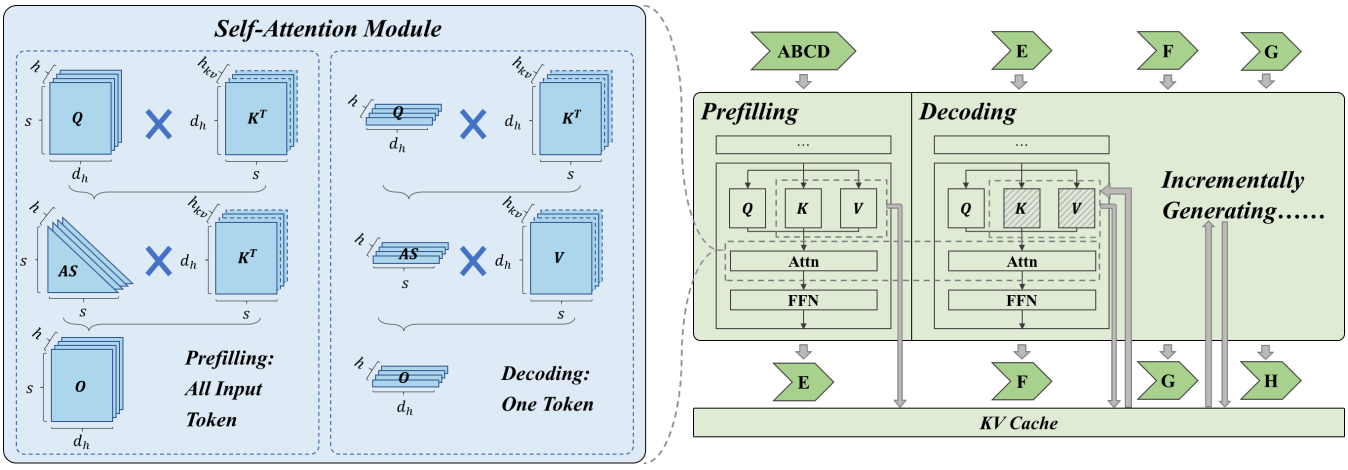

FIGURE 2 Overview of LLM Inference. The left part illustrates the computation process of the self-attention module, where "Q","K","V","AS", and "O" represent query, key, value, attention score, and output, respectively. The right part depicts the LLM inference process, consisting of the prefilling phase and the decoding phase, where "Attn" and "FFN" represent the attention layer and the feed-forward network layer, respectively. The mathematical notations are detailed in Table 1.

2 | BACKGROUND

In this section, we discuss the fundamental concepts related to large language model (LLM) inference, current techniques in LLM quantization, and recent research on KV Cache compression.

TABLE 1 Notations for LLM inference mechanisms explanation and method formula. This table lists key symbols and their meanings used in Sec 2.1 and Sec 3.

Sym.	Explanation	Sym.	Explanation
n	Batch Size	h	The number of heads
s	Current Sequence Length	C	The number of channels.
d	The dimensionality of hidden states	l	The number of layers
h_{kv}	The number of Heads (for keys and values)	d_h	The dimensionality of each head
$Q(\cdot)$	Quantization Operator	$\boldsymbol{X}$	Activation or Weight
$\boldsymbol{S_X}$	Scaling Factor	$\boldsymbol{Z_X}$	Integer Zero Point
B	Quantization Bit Width	G	Group Size
N	Window Size	F	First Few Token Size

2.1 | LLM Inference

An overview of LLM inference is depicted in Figure 2. LLM comprises a stack of transformer layers, along with a vocabulary embedding for input and a token classifier for output. The self-attention module, which is a crucial component of a transformer layer, facilitates interaction and information aggregation among different tokens.

Multi-Head Attention (MHA) and Grouped-Query Attention (GQA)[31] are the primary variants of the self-attention module. Following the notations in Table 1, the attention module receives an input of shape (n, s, d). In MHA, the input is separately projected and transposed for query, key, and value, resulting in the same shape of (n, h, s, d_h), where it usually holds that $d = h \cdot d_h$. Different heads are expected to capture different semantic information. The attention mechanism multiplies the queries and keys, applies a lower-triangular mask to restrict queries to preceding keys only, and performs softmax to obtain the attention scores of shape (n, h, s, s). The attention scores are then used to weighted-sum the values, yielding an output of shape (n, h, s, d_h), which is later reshaped into (n, s, d). To alleviate memory and computation burden, GQA employs a smaller number of heads h_{kv} for keys and values, resulting in their shape being (n, h_{kv}, s, d_h). In this setup, each key-value pair corresponds to multiple queries.

During LLM inference, the model operates in an auto-regressive manner, generating one token at a time in sequential execution. As illustrated in Figure 2, the process can be divided into two phases: prefilling and decoding. In the prefilling phase, the self-attention module processes the entire input sequence, computing queries, keys, and values for all tokens. The resulting key-value pairs are stored in the Key-Value (KV) cache for subsequent use. In the decoding phase, the model operates auto-regressively, generating tokens one at a time. Here, the attention module computes the query, key, and value only for the most recently generated token, while leveraging previously stored keys and values from the KV Cache to compute the attention scores of shape $(n, h, 1, s)$. The newly generated key-value pairs are then added to the KV Cache. The size of the KV Cache can be calculated as $(n \times s \times l \times 2 \times h \times d_h)$ floating-point numbers, where n is the batch size, s is the current sequence length, l is the number of layers and the factor of 2 accounts for separate caching of keys and values.

As illustrated in Figure 1, the contribution of the KV Cache to memory consumption becomes increasingly significant, particularly for long sequence lengths and larger batch sizes. This linear scaling of memory consumption concerning sequence length and batch size creates a substantial bottleneck in scenarios involving long-context LLM inference.

2.2 | LLM Quantization

There are many quantization methods designed for LLMs. One primary approach is weight-only quantization, which reduces the precision of model weights by representing them with lower-bit formats, such as 4-bit or 8-bit integers. For example, SPQR[32] strategically identifies and decomposes the model's weight matrix into dense and sparse components. The sparse part, characterized by numerous outliers, is retained in high precision, while the dense portion undergoes quantization and compression to preserve accuracy. GPTQ[33] employs second-order approximation techniques to quantize weights, enabling the

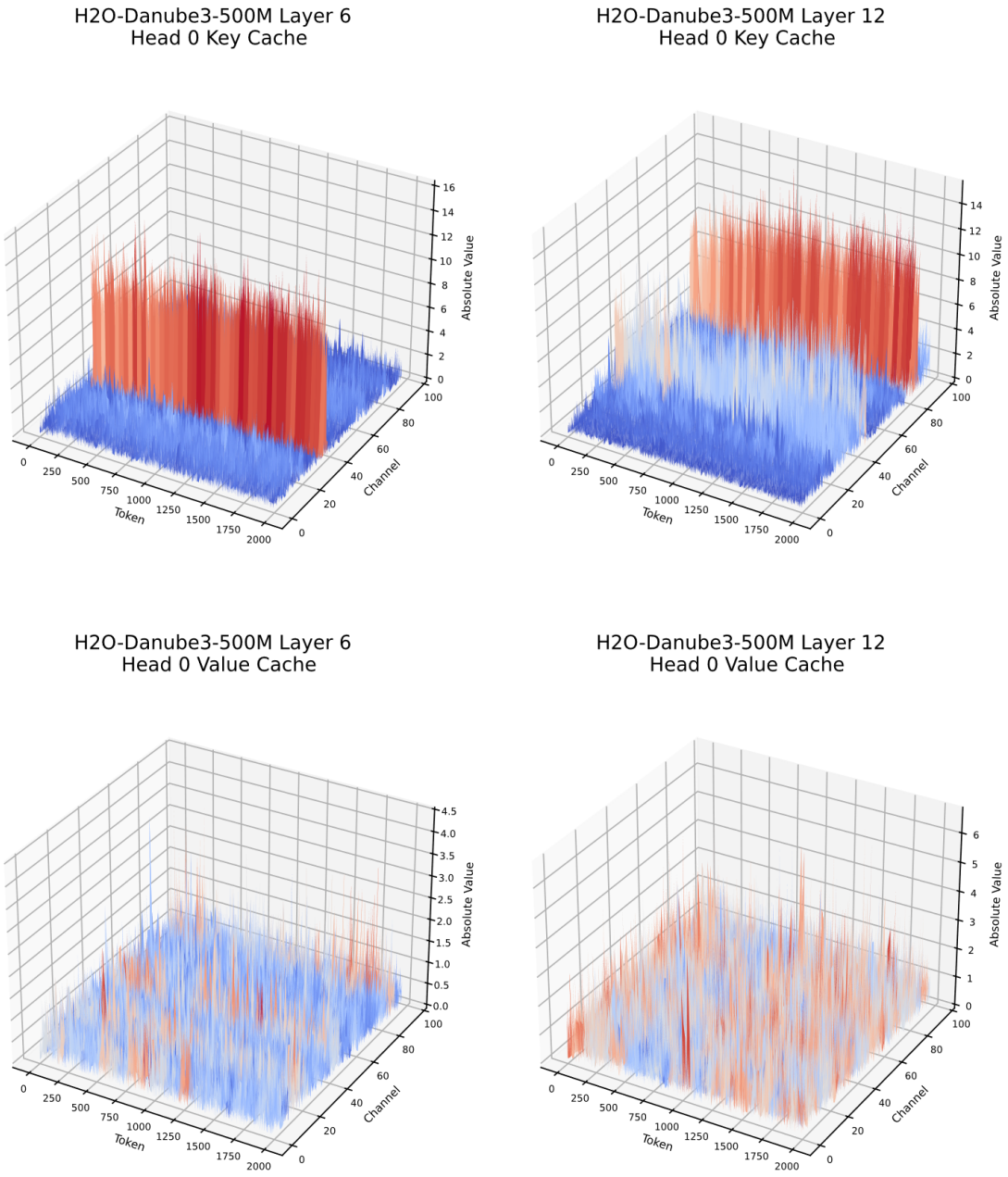

FIGURE 3 Magnitude of key and value cache for H2O-Danube3-500M. We observe (1) for key cache, there are a few channels whose magnitudes are very large. (2) for value cache, there is no obvious outlier pattern.

compression of LLMs into 4-bit precision. Similarly, AWQ[34] applies activation-aware quantization to achieve 4-bit weight precision. Furthermore, SqueezeLLM[35] innovatively introduces sensitivity-based non-uniform quantization in conjunction with Dense-and-Sparse decomposition, allowing it to adapt to the varying distribution and sensitivity of weights across different channels. While these techniques provide significant compression, they are orthogonal to our work and can be used in conjunction with our methods.

Another line of work focuses on weight-activation quantization, which involves quantizing model weights and activations. Methods such as llm.int8()[22] retain outlier channels at full precision, enabling other components to be compressed to 8-bit. OmniQuant[36] enhances the calibration dataset by utilizing synthetic data and perturbation techniques to more accurately estimate quantization parameters. This approach allows for a more robust calibration process, improving the overall fidelity of the quantization. In contrast, SmoothQuant[37] addresses quantization errors by employing activation value scaling in conjunction with weight inverse scaling. This technique effectively smooths out outliers in the activation values, thereby reducing quantization error and enhancing the precision of the quantized model. RPTQ[10] reorders channels to reduce variance within quantization clusters, enhancing accuracy. ATOM[38] improves quantization performance and reduces inference latency through finer-grained quantization with efficient kernels. These methods are not specifically optimized for KV cache compression. Despite exhibiting strong empirical performance in alternative settings, their application still incurs substantial limitations in KV Cache compression.

Recently, as natural language tasks require processing longer contexts, researchers have focused on quantizing KV Cache. Several new methods designed for multi-billion scale LLMs have been developed, such as KVQuant[20], WKVQuant[21], KIVI[19] and SKVQ[30]. Both KVQuant[20] and KIVI[19] adopt quantization methods that apply per-channel quantization to the key cache and per-token quantization to the value cache. However, they differ significantly in their implementations. KVQuant[20] relies on offline calibration data to derive channel quantization parameters for the key cache and uses non-uniform quantization for KV Cache processing. In contrast, KIVI[19] introduces a fixed quantization window, only quantizing tokens outside the window. WKVQuant[21] adopts a different approach by quantizing only the past KV Cache while maintaining full precision for the key-value pairs in the current step. Its key innovation lies in a technique called two-dimensional quantization, which includes static channel smoothing to reduce inter-channel value differences and dynamic token quantization to compute quantization parameters for each token dynamically. On the other hand, SKVQ[30] employs offline calibration to derive a channel reorder matrix for both Key and Value. This approach enables per-token quantization of the KV Cache while also incorporating a fixed quantization window to enhance efficiency and precision.

These methods represent significant advancements in optimizing KV Cache quantization to improve performance in long-context scenarios. However, several limitations remain. For instance, the fixed quantization window employed in methods such as KIVI[19], WKVQuant[21], and SKVQ[30] does not yet support dynamic expansion for fully precise quantization windows, limiting its flexibility in adapting to varying input contexts. Additionally, KIVI[19] and WKVQuant[21] do not incorporate Attention Sink-Aware Quantization, while SKVQ[30] applies a per-token quantization approach similar to RPTQ[10]. Moreover, as KVQuant[20] relies on offline calibration data to derive key cache channel quantization parameters, its quantization effectiveness is adversely affected. A key innovation of SubKV is its integration of per-channel quantization, attention sink-aware quantization, and dynamic quantization window into a unified quantization framework. Our experimental results demonstrate that our method outperforms existing approaches in long-context tasks within sub-billion parameter LLMs.

2.3 | KV Cache Compression

There have also been several prior works on compressing the KV Cache. Some of these methods aim to only store important tokens in the KV Cache and to evict less important tokens, thereby maintaining low memory usage[17,25,39,26,40]. Adaptive KV Cache Compression[41] leverages the self-attention mechanism of the model to compute the attention weights between the current generation position and all previous positions. This allows for a selective discarding of the corresponding KV Cache, optimizing memory usage by retaining only the most relevant key-value pairs. Similarly, Scissorhands[42] operates under the assumption that the importance of tokens within an input sequence maintains a level of persistence across different layers. By utilizing the model's attention mechanism, it evaluates and subsequently expels the KV Cache associated with less important tokens, thereby streamlining memory consumption. Another noteworthy approach is SparQ Attention[43], which selectively retrieves key historical information by transmitting only the KV Cache with high attention scores. This method significantly reduces the demands on memory bandwidth, further enhancing the efficiency of LLMs during inference.

Other methods aim to only retrieve a subset of tokens at each step to achieve memory bandwidth savings[43]. Unlike KV Cache quantization, which retains all cache but compresses them to low precision, these methods selectively retain part of the KV Cache and discard other cache directly. These methods usually allocate a fixed-size buffer for KV Cache. When the generated KV Cache exceeds the buffer limit, some tokens considered less important will be evicted from the buffer. In this work, we explore KV Cache quantization as an orthogonal direction for compressing the KV Cache to enable long context inference.

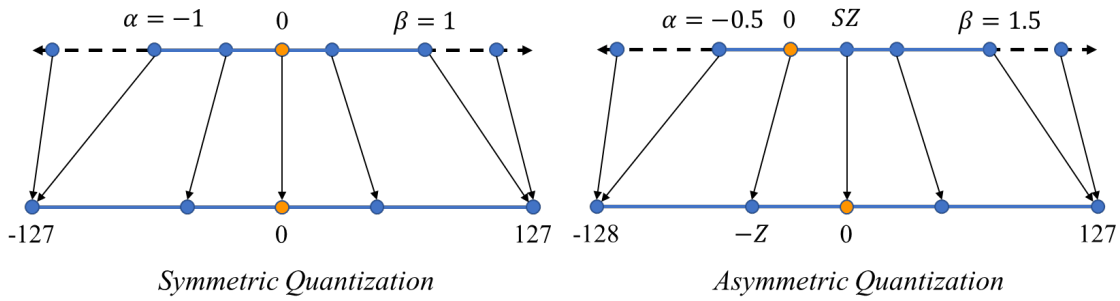

FIGURE 4 Illustration of symmetric quantization and asymmetric quantization. Symmetric quantization with restricted range maps real values to [-127, 127], and full range maps to [-128, 127] for 8-bit quantization.

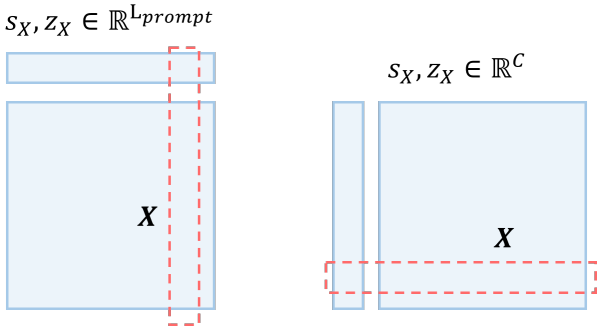

FIGURE 5 Definition of per-token and per-channel quantization. Let $X \in \mathbb{R}^{L_{\text{prompt}} \times C}$ be the key/value cache, where L_{prompt} is the number of tokens and C is the number of channels. Let $\boldsymbol{Z_X}$ be the integer zero-point, and $\boldsymbol{S_X}$ be the scaling factor.

Some existing research has attempted to address KV Cache issues from a systems perspective[44]. For instance, Sheng et al.[45] integrated GPU, CPU, and disk memory and computational resources to offload the KV Cache to non-GPU memory, alleviating memory constraints. Similarly, Kwon et al.[46] and Jin et al.[47] extended virtual memory and paging techniques to the attention mechanism, effectively reducing the memory requirements of the KV Cache while enhancing model throughput. This research direction is orthogonal to our work since system-level optimizations can also be applied to our algorithm.

3 | THE PROPOSED METHOD

In scenarios involving long contexts or batched inferences, the memory and computational bottlenecks are primarily associated with the storage and loading of the KV Cache. A simple and effective solution to mitigate this issue is to reduce the memory footprint of the KV Cache, with quantization serving as a key technique. To address this, we first introduce the foundational concepts of KV Cache quantization in Section 3.1. In Section 3.2, we explore the rationale behind quantizing the key and value cache along distinct dimensions. Section 3.3 and Section 3.4 investigate how leveraging the phenomenon of attention locality can enhance the performance of quantized models. Building upon these insights, we propose SubKV, a novel KV Cache quantization method, which is elaborated in Section 3.5.

3.1 | Preliminary Study of KV Cache Quantization

In this part, we briefly introduce the basic quantization methods. First, We need to define a function that can quantize neural network weights and activations to a finite set of values. This function takes real values in floating point and maps them to a

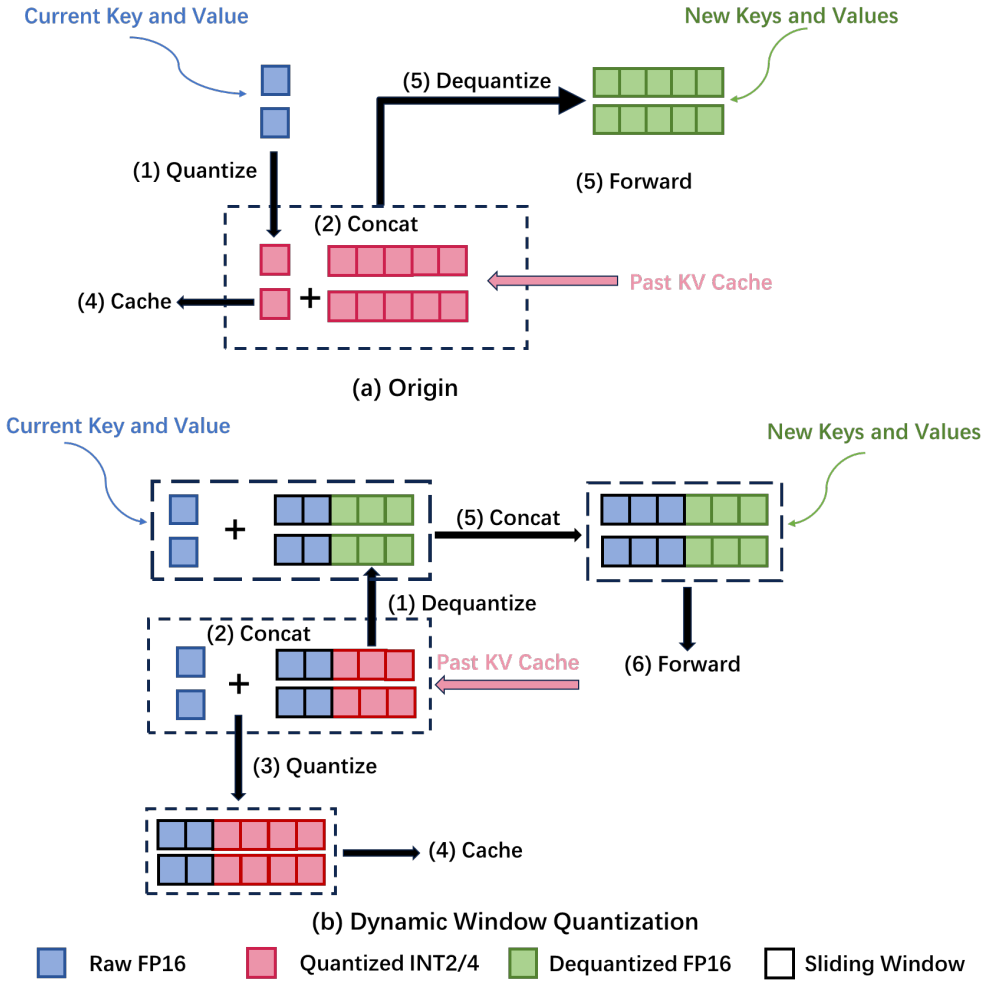

FIGURE 6 Comparison between original quantization method and Dynamic Window Quantization (DWQ) for KV Cache. DWQ selectively quantizes only the KV Cache for tokens falling outside the defined window length, provided that the attention score of the edge token is lower than the attention score of the newly introduced token. In this context, FP16 refers to 16-bit floating-point representation, which allows for reduced memory usage while maintaining numerical precision. Int2 and Int4 denote 2-bit and 4-bit integer quantizations.

lower precision range. A popular choice for a quantization function is as follows:

$$Q(\boldsymbol{X}) = \frac{\boldsymbol{X}}{\boldsymbol{S}_X} - \boldsymbol{Z}_X, \tag{1}$$

where $Q(\cdot)$ is the quantization operator, $\boldsymbol{X}$ is a real valued input (activation or weight), $\boldsymbol{S}_X$ is a scaling factor value, and $\boldsymbol{Z}_X$ is an integer zero point. Furthermore, the Int function maps a real value to an integer value through a rounding operation (e.g., round to nearest and truncation). Essentially, this function maps real values $\boldsymbol{X}$ to some integer values. This method of quantization is also known as uniform quantization, as the resulting quantized values. It is possible to recover real values $\boldsymbol{X}$ from the quantized values $Q(\boldsymbol{X})$ through an operation that is often referred to as dequantization:

$$\tilde{\boldsymbol{X}} = \boldsymbol{S}_X(Q(\boldsymbol{X}) + \boldsymbol{Z}_X), \tag{2}$$

Note that the recovered real values $\tilde{\boldsymbol{X}}$ will not exactly match $\boldsymbol{X}$ due to the rounding operation. One important factor in uniform quantization is the choice of the scaling factor $\boldsymbol{Z}_X$ in Eq. 1. This scaling factor essentially divides a given range of real values

$\boldsymbol{X}$ into several partitions:

$$\boldsymbol{S_X} = \frac{\beta - \alpha}{2^B - 1}, \tag{3}$$

where $[\alpha, \beta]$ denotes the clipping range, a bounded range that we are clipping the real values with, and B is the quantization bit width. Therefore, for the scaling factor to be defined, the clipping range $[\alpha, \beta]$ should first be determined.

The process of choosing the clipping range is often referred to as calibration. A straightforward choice is to use the min/max of the signal for the clipping range, i.e., $\alpha = \min(\boldsymbol{X})$, and $\beta = \max(\boldsymbol{X})$. This approach is an asymmetric quantization scheme, since the clipping range is not necessarily symmetric concerning the origin, i.e., $-\alpha \neq \beta$, as illustrated in Figure 4. It is also possible to use a symmetric quantization scheme by choosing a symmetric clipping range of $\alpha = -\beta$. A popular choice is to choose these based on the min/max values of the signal: $-\alpha = \beta = \max(|\max(\boldsymbol{X})|, |\min(\boldsymbol{X})|)$. Asymmetric quantization often results in a tighter clipping range as compared to symmetric quantization. This is especially important when the target weights or activations are imbalanced, e.g., the activation after ReLU that always has non-negative values. Using symmetric quantization, however, simplifies the quantization function in Eq. 1 by replacing the zero point with $\boldsymbol{Z_X} = 0$.

Given that KV Cache quantization involves model weights and activation values, symmetric quantization is suboptimal in scenarios where the distribution is skewed and asymmetric. Therefore, we opt for asymmetric quantization in our approach. To the best of our knowledge, one of the most flexible methods for quantizing the KV Cache is through round-to-nearest quantization. This process for B-bit integer quantization and dequantization can be succinctly expressed as follows:

$$Q(\boldsymbol{X}) = \left\lfloor \frac{\boldsymbol{X} - \boldsymbol{Z_X}}{\boldsymbol{S_X}} \right\rceil, \quad \tilde{\boldsymbol{X}} = Q(\boldsymbol{X}) \cdot \boldsymbol{S_X} + \boldsymbol{Z_X}, \tag{4}$$

where $\boldsymbol{Z_X} = \min(\boldsymbol{X})$ is the zero-point, B is the quantization bit width, $\boldsymbol{S_X} = (\max(\boldsymbol{X}) - \min(\boldsymbol{X}))/(2^B - 1)$ is the scaling factor, and $\lfloor \cdot \rceil$ is the rounding operation. Here we ignore the batch size for ease of understanding. As shown in Figure 5, $\boldsymbol{X}$ is quantized along either the token or channel dimension group-wisely.

3.2 | Per-Channel Key Quantization

In our proposed approach, we first perform a detailed analysis to understand the KV Cache distributions. Figure 3 shows sample distributions for the KV Cache activations. We observe that in the key cache, some fixed channels exhibit very large magnitudes, whereas in the value cache, there is no significant pattern for outliers. This finding aligns with prior research that highlights the presence of outlier channels in the activations of large language models [18,19,20,22]. The presence of outliers in each channel means that per-channel quantization confines the error to the respective channel, without affecting the other normal channels. Furthermore, the value matrices not only display outlier channels but also exhibit outlier tokens, although these outlier tokens are generally less pronounced compared to their counterparts in the key channels.

Many existing approaches to KV Cache quantization have employed per-token quantization techniques, wherein the scaling factor and zero-point are uniformly shared among the elements within the same token. [38,45]. However, due to the differing average magnitudes between channels, the values within a channel are easier to quantize when grouped than the values across different channels. We investigate the application of per-channel Key quantization in sub-billion parameter LLMs. This approach implies that the scaling factor and zero-point are shared among elements within the same channel. By sharing the scaling factor and zero-point along the channel dimension, this will naturally group together values with similar magnitudes, thereby mitigating the impacts of outlier channels on other channels when quantizing to low precision. The per-token value cache quantization aligns well with the streaming nature of auto-regressive inference, allowing newly quantized tensors to be directly appended to the existing quantized value cache by token dimension. However, for per-channel key quantization, the quantization process spans across different tokens, which cannot be directly implemented in the streaming setting. Our solution is to group the key cache of every G token and quantize them separately. A detailed explanation of this approach is provided in Section 3.5.

3.3 | Dynamic Window Quantization

In autoregressive token generation, each layer's key and value cache are typically stored in a KV Cache, which serves as input for subsequent token generation, thereby reducing redundant computations across steps. This optimization, while computationally beneficial for LLMs, leads to increased memory consumption, particularly when processing large batches or lengthy input

sequences (refer to Figure 1). Furthermore, as illustrated in Figure 6(a), existing quantization methods require quantizing and dequantizing the current keys and values before inference, introducing computational overhead through redundant precision conversions.

Prior research demonstrates strong locality patterns in attention modules[23,24,25], where recent tokens receive disproportionately high attention weights at each timestep. We emphasize that compared to a large amount of but less important content in the previous KV Cache, protecting the accuracy of these small portions of but more important cache in KV Cache quantization is critical. Motivated by this, we proposed a Dynamic Window Quantization Strategy (DWQ), which retains the latest KV Cache of a window size w tokens to full precision. 1) In the prefill phase, for each transformer block, after the KV Cache is generated, we first compute attention with full precision KV Cache, then quantize the KV Cache with the last window size token cache pairs reserved as full precision. 2) In the decode phase, we only process the token which slides out of the window at each time step. As shown in Figure 6(b), DWQ does not quantize the representations generated from the current token. Instead, it uses the original full precision representations when merging them with the dequantized past KV Cache during the forward process. For quantizing the tokens within the window, our criterion is as follows: if the attention scores of newly generated tokens are less than those of the window's edge tokens, we opt to expand the window size to retain the important edge tokens; conversely, if the attention scores of newly generated tokens exceed those of the edge tokens, we proceed to quantize the KV Cache of the window's edge tokens. This approach ensures the KV Cache of each transformer block generated in the prefill phase is lossless. It also enhances the quality of generated content by utilizing the locality of the attention module in the decode phase.

3.4 | Attention Sink-Aware Quantization

Previous studies have shown that in large language models (LLMs), a significant attention score is disproportionately allocated to the first few tokens after the initial few layers[26]. This phenomenon occurs even when the first few token lacks semantic significance, as the model tends to utilize it as an attention sink. We observed it is effective to keep a small number of sink tokens with high precision. Since the positions of sink tokens are fixed, it is easy to implement and we enable it in our experiments. In our work, we demonstrate that, due to the attention sink phenomenon, the model exhibits heightened sensitivity to quantization errors associated with the first token. By representing only the first token in FP16 format, we achieve notable perplexity reductions, particularly in the context of 2-bit quantization (Table 5). This observation aligns with findings from other recent studies[48,20].

3.5 | SubKV Algorithm

In this section, we elaborate on the algorithm for SubKV. We highlighted in Section 3.2 that the quantization process, which spans across different tokens, cannot be directly implemented in a streaming setting. To address the challenges associated with per-channel quantization, our primary strategy to resolve this issue involves grouping the key cache of every G token and performing separate quantization for each group. This approach not only facilitates efficient processing within the constraints of streaming environments but also enhances the granularity of the quantization, thereby optimizing the representation of the cache and improving overall model performance. Because the number of tokens in X_K can be arbitrary, we split X_K into three parts, namely, the first few token parts $X_{K_f} = X_K[: F]$, the grouped part $X_{K_g} = X_K[F : L-N]$, and the window residual part $X_{K_n} = X_K[L-N-F :]$, where L is the number of tokens inside the current key cache X_K, N is the number of window residual tokens, and $(L-N-F)$ is divisible by G.

Since X_{K_g} can be evenly divided into $(L-N-F)/G$ groups, we only store $Q(X_{K_g})$ with group-wise quantization, while X_{K_n} and X_{K_f} are kept in full precision. During the decoding process, each newly arrived key cache t_K is added to X_{K_n}. With tiled matrix multiplication, the raw attention logits are then calculated as:

$$\boldsymbol{A}_g = \boldsymbol{t}_Q Q\left(\boldsymbol{X}_{K_g}^{\top}\right), \tag{5}$$

$$\boldsymbol{X}_{K_n} = \text{Concat}\left(\left[\boldsymbol{X}_{K_n}, \boldsymbol{t}_K\right]\right), \tag{6}$$

$$\boldsymbol{A}_n = \boldsymbol{t}_Q \boldsymbol{X}_{K_n}^{\top}, \tag{7}$$

$$\boldsymbol{A}_f = \boldsymbol{t}_Q \boldsymbol{X}_{K_f}^\top, \tag{8}$$

$$\boldsymbol{A} = \text{Concat}\left(\left[\boldsymbol{A}_f, \boldsymbol{A}_g, \boldsymbol{A}_n\right]\right). \tag{9}$$

Once X_{K_n} reaches N tokens, defined as the hyperparameter representing the window residual length, we will compare the attention score of the newly introduced token, $A_{\text{new}} = \text{Softmax}(A)[-1]$, with the attention score of the edge token, $A_{\text{edge}} = \text{Softmax}(A)[-N]$. If $A_{\text{new}} > A_{\text{edge}}$, we proceed to quantize the edge token and concatenate it with the previously quantized $Q(X_{K_g})$; conversely, if $A_{\text{new}} \leq A_{\text{edge}}$, we extend the window length to $N + 1$. This adaptive mechanism not only ensures that we retain the most relevant tokens based on their attention scores but also optimizes the utilization of the key-value cache, thereby enhancing the efficiency of the quantization process within the streaming framework.

For the value cache, analogous to the key cache, we also partition it into three segments, maintaining the most recent value cache in full precision, namely X_{V_f}, X_{V_g}, and X_{V_n}. Specifically, we employ a queue structure, where each newly arrived value cache is pushed into the queue. Once the queue reaches the predefined window window residual length N, the oldest value cache is removed from the queue. Subsequently, the evicted value cache undergoes per-token quantization and is concatenated with the previously quantized value cache along the token dimension.

It is noteworthy that the order of quantization is such that the key cache is quantized first, followed by the value cache. This means that if, during the quantization of the key cache, SubKV opts to extend the window length instead of quantizing the key cache, the value cache will similarly remain unquantized due to the failure to meet the condition of the queue reaching the predefined window residual length N. Conversely, if the key cache is quantized, the quantization of the value cache will inevitably follow, ensuring synchronization between these two caches and optimizing the overall quantization process. Specifically, we provide the pseudocode for SubKV when calculating the attention output in the prefill and decoding phases (Algorithm 1).

4 | EXPERIMENTS

We conducted several experiments to evaluate the effectiveness of the SubKV method in quantizing the KV Cache for sub-billion parameter LLMs. In this section, we introduce the models used, the experimental settings, and the experiment results obtained from our evaluations.

4.1 | Settings

Models. We evaluated the SubKV using four sub-billion parameter models: H2O-Danube3-500M[7], MobileLLM-600M[6], MobileLLM-1B[6], and TinyLlama[27]. The selection of these models is driven by the observation that existing KV Cache quantization methods have predominantly targeted large models built upon the Llama architecture. To enable meaningful comparisons with other quantization techniques, we thus restrict our analysis to sub-billion parameter LLMs that are trained on the Llama architecture.

- **H2O-Danube3-500M** is a decoder-only language model built on the Llama architecture, incorporating advancements from Llama 2[2] and Mistral[49]. It uses the Mistral tokenizer with a vocabulary size of 32,000 and supports a context length of up to 8,192 tokens. The model features Grouped Query Attention[31], enhancing parameter efficiency and computational performance. Trained on 4 trillion tokens, it is well-suited for diverse language processing tasks.
- **TinyLlama** is a decoder-only language model based on the Llama architecture and trained on 1 trillion tokens. It also adopts Grouped Query Attention[31], effectively reducing memory bandwidth overhead while maintaining strong performance.
- **MobileLLM**, developed by Meta, integrates several key optimizations: (1) SwiGLU activation function[50], (2) deep and thin architectures, (3) embedding sharing[3], and (4) Grouped Query Attention[31]. These design choices are aimed at maximizing efficiency and performance.

We implemented the SubKV on the Hugging Face Transformers codebase and conducted all experiments on a single NVIDIA RTX 3090 GPU (24GB).

Algorithm 1 The SubKV Prefill & Decoding Algorithm

Parameter: group size G, window length N, first few token FP16 F
procedure PREFILL
  Input: $X \in \mathbb{R}^{L_{\text{prompt}} \times C}$
  $X_K \leftarrow XW_K, \quad X_V \leftarrow XW_V$
  $X_{V_f} \leftarrow X_V[:F], \quad X_{K_f} \leftarrow X_K[:F]$
  $X_V \leftarrow X_V[F:], \quad X_K \leftarrow X_K[F:]$
  $X_{V_g} \leftarrow X_V[:L_{\text{prompt}} - N], \quad X_{V_n} \leftarrow X_V[L_{\text{prompt}} - N:]$
  $Q(X_{V_g}) \leftarrow \texttt{GroupQuant}(X_{V_g}, \texttt{dim=token}, \texttt{numGroup} = d//N)$
  $Q(X_{K_g}), \quad X_{K_n} \leftarrow \texttt{KeyQuant}(X_K)$
  KV Cache $\leftarrow Q(X_{K_g}), X_{K_n}, Q(X_{V_g}), X_{V_n}, X_{V_f}, X_{K_f}$
  return X_K, X_V
end procedure
procedure DECODING
  Input: KV Cache, $t \in \mathbb{R}^{1 \times C}$
  $t_Q \leftarrow tW_Q, \quad t_K \leftarrow tW_K, \quad t_V \leftarrow tW_V$
  $Q(X_{K_g}), X_{K_n}, Q(X_{V_g}), X_{V_n} \leftarrow$ KV Cache
  $X_{K_n} \leftarrow \texttt{Concat}([X_{K_n}, t_K], \texttt{dim=token})$
  $X_{V_n} \leftarrow \texttt{Concat}([X_{V_n}, t_V], \texttt{dim=token})$
  $A \leftarrow \texttt{Concat}([t_Q Q(X_{K_g})^\top, t_Q (X_{K_n})^\top, t_Q (X_{K_f})^\top], \texttt{dim=token})$
  $A_g \leftarrow \texttt{Softmax}(A)[F:-N], \quad A_n \leftarrow \texttt{Softmax}(A)[-N:], \quad A_f \leftarrow \texttt{Softmax}(A)[:F]$
  if $\text{len}(X_{K_n}) = N$ **then**
    if $A_n[-N] > A_n[1]$ **then**
      $N \leftarrow N + 1$
    else
      $Q(X_{K_n}), \_ \leftarrow \texttt{KeyQuant}(X_{K_n})$
      $Q(X_{K_g}) \leftarrow \texttt{Concat}([Q(X_{K_g}), Q(X_{K_n})], \texttt{dim=token})$
      $X_{K_n} \leftarrow$ empty tensor of shape $(0, C)$
    end if
  end if
  if $\text{len}(X_{V_n}) > N$ **then**
    $Q(X_{V_n'}) \leftarrow \texttt{GroupQuant}(X_{V_r}[:-N], \texttt{dim=token}, \texttt{numGroup} = C//G)$
    $Q(X_{V_g}) \leftarrow \texttt{Concat}([Q(X_{V_g}), Q(X_{V_n'})], \texttt{dim=token})$
    $X_{V_n} \leftarrow X_{V_r}[-N:]$
  end if
  $t_O \leftarrow A_g Q(X_{V_g}) + A_n X_{V_n} + A_f X_{V_f}$
  KV Cache $\leftarrow Q(X_{K_g}), X_{K_n}, Q(X_{V_g}), X_{V_n}, X_{V_f}, X_{K_f}$
  return t_O
end procedure
function KEYQUANT($X_K \in \mathbb{R}^{L \times C}$)
  $N \leftarrow L \mod N$
  $X_{K_g} \leftarrow X_K[:L-N]$
  $X_{K_n} \leftarrow X_K[L-N:]$
  $Q(X_{K_g}) \leftarrow \texttt{GroupQuant}(X_{K_g}, \texttt{dim=channel}, \texttt{numGroup} = \lfloor L/G \rfloor)$
  return $Q(X_{K_g}), X_{K_n}$
end function

TABLE 2 Description of the used LongBench datasets. This table provides an overview of the datasets utilized in the study, including their tasks, types, evaluation metrics, average lengths, languages, and sample sizes.

Dataset	Task Type	Metric	Avg. Length	Language	Samples
Qasper[51]	Single-Document QA	F1 score	3,619	EN	200
QMSum[52]	Summarization	Rouge-L	10,614	EN	200
MultiNews[53]	Summarization	Rouge-L	2,113	EN	200
TREC[54,55]	Few-shot Learning	Accuracy	5,177	EN	200
TriviaQA[56]	Few-shot Learning	F1 score	8,209	EN	200
SAMSum[57]	Few-shot Learning	Rouge-L	6,258	EN	200
LCC[58]	Code Completion	Edit Sim	1,235	Python/C#/Java	500
RepoBench-P[59]	Code Completion	Edit Sim	4,206	Python/Java	500

Task. As discussed in Section 2, the KV Cache size grows significantly with longer context lengths. We evaluate SubKV under a long-context setting using generation tasks from LongBench[60]. Specifically, we adopt tasks from four subgroups of LongBench. Detailed dataset information is summarized in Table 2. We also consider the perplexity (PPL) evaluation using Wikitext2[28] and C4[29].

- **Single-Document QA:** Qasper[51] (F1 score).
- **Summarization:** QMSum[52] and MultiNews[53] (Rouge-L).
- **Few-shot Learning:** TREC[54,55] (Accuracy), TriviaQA[56] (F1 score), and SAMSum[57] (Rouge-L).
- **Code Completion:** LCC[58] and RepoBench-P[59] (Edit Sim).

For LongBench, task-specific metrics are used, and the overall performance is reported as the average score across datasets. The maximum sequence length is set to 8192 for H2O-Danube3-500M and 2048 for MobileLLM and TinyLlama, aligning with the respective maximum context lengths of these models. For PPL evaluation, the maximum sequence length is uniformly set to 2048 across all models.

Baseline. For LongBench, we include the FP16 baseline, SubKV-4bit, and SubKV-2bit. For PPL evaluation, we compare against the FP16 baseline, KVQuant[20], KIVI[19], and SKVQ[30]. The group size for all methods is fixed at 32. SubKV uses a window size of 32, and KIVI[19] is configured with a residual length of 32.

4.2 | Main Results

In this section, we present a comprehensive analysis of the experimental results and the effectiveness of the proposed SubKV method. The experiments include assessments on LongBench, Perplexity, Long Context Evaluation, and Efficiency Comparison. These analyses demonstrate SubKV's effectiveness in maintaining performance across various tasks while significantly reducing memory requirements.

4.2.1 | LongBench results

The performance of SubKV on H2O-Danube3-500M, MobileLLM-600M, MobileLLM-1B and TinyLlama-1.1B-chat models in the LongBench dataset is summarised in Table 3. Table 3 suggests that SubKV is an effective method for KV Cache compression with minimal impact on accuracy across various hard long context generation tasks. For instance, when the KV Cache of the H2O-Danube3-500M model is quantized to 4 bits, its average score on LongBench remains comparable to that of the FP16 baseline model. Furthermore, quantizing the KV Cache to 2 bits results in an average score that decreases by only 0.58 relative to the baseline. A similar trend is observed with other models. When the KV Cache of H2O-Danube3-500M is quantized to 4 bits, the average score experiences a negligible decline. Although there is a drop of 1.52 in average score when the KV Cache is quantized to 2 bits, the model retains robust performance even at extremely low quantization precision.

TABLE 3 Performance evaluation of SubKV on H2O-Danube3-500M, MobileLLM-600M, MobileLLM-1B and TinyLlama-1.1B-chat models across a range of benchmarks in LongBench. We highlight the average performance of our method.

Model	Methed	LongBench Score ↑								
		Qasper	QMSum	MultiNews	TREC	TriviaQA	SAMSum	LCC	RepoBench-P	Avg.
H2O-Danube3-500M	FP16	13.36	18.74	12.79	66.50	78.99	34.27	20.17	28.51	34.16
	SubKV-4bit	14.08	18.87	12.79	66.50	79.56	34.04	20.37	28.41	**34.33**
	SubKV-2bit	13.92	19.04	13.80	66.50	76.91	33.08	19.46	25.63	**33.58**
MobileLLM-600M	FP16	5.81	15.10	5.94	46.00	52.37	35.67	50.37	47.75	32.38
	SubKV-4bit	6.42	15.18	6.37	45.50	51.92	35.91	50.29	47.57	**32.39**
	SubKV-2bit	5.49	13.78	4.62	45.00	51.37	34.38	48.49	46.99	**31.27**
MobileLLM-1B	FP16	6.36	17.21	13.21	44.50	60.80	37.97	57.26	49.86	35.90
	SubKV-4bit	6.43	17.45	13.05	44.50	60.82	37.96	56.29	50.31	**35.85**
	SubKV-2bit	6.28	16.68	8.94	44.50	60.94	37.53	53.40	48.41	**34.59**
TinyLlama-1.1B-chat	FP16	9.67	19.54	20.41	52.5	54.56	33.64	53.35	45.4	36.13
	SubKV-4bit	9.58	19.41	20.26	53.5	53.78	32.84	53.57	45.68	**36.08**
	SubKV-2bit	8.58	17.73	18.27	51.0	54.85	32.82	49.91	43.71	**34.61**

TABLE 4 Performance comparison of H2O-Danube3, MobileLLM-600M, MobileLLM-1B and TinyLlama-1.1B-chat models on Wikitext2 and C4 Datasets. This table presents the perplexity results for both models with a maximum sequence length of 2048 tokens. All methods utilize a uniform group size of 32, while the window size for SubKV is also set to 32, ensuring consistency in evaluation. Additionally, the residual length in KIVI[19] is configured to 32, facilitating a comparison of model performance.

Model		Method	Perplexity ↓		Model		Method	Perplexity ↓	
			WikiText2	C4				WikiText2	C4
H2O-Danube3-500M	FP16	-	11.68	13.65	MobileLLM-1B	FP16	-	8.97	6.30
	4-bit	KVQuant[20]	11.79	13.73		4-bit	KVQuant[20]	9.07	6.34
		KIVI[19]	11.75	13.68			KIVI[19]	9.01	6.32
		SKVQ[30]	11.95	13.70			SKVQ[30]	9.09	6.39
		SubKV	**11.69**	**13.67**			SubKV	**9.00**	**6.31**
	2-bit	KVQuant[20]	17.58	17.00		2-bit	KVQuant[20]	10.35	7.57
		KIVI[19]	13.42	14.70			KIVI[19]	9.78	7.04
		SKVQ[30]	14.19	14.56			SKVQ[30]	9.47	6.76
		SubKV	**12.27**	**14.02**			SubKV	**9.35**	**6.70**
MobileLLM-600M	FP16	-	10.10	6.57	TinyLlama-1.1B-chat	FP16	-	9.48	6.39
	4-bit	KVQuant[20]	10.23	6.62		4-bit	KVQuant[20]	9.63	6.43
		KIVI[19]	10.15	6.59			KIVI[19]	9.52	6.42
		SKVQ[30]	10.27	6.63			SKVQ[30]	9.66	6.54
		SubKV	**10.13**	**6.58**			SubKV	**9.51**	**6.41**
	2-bit	KVQuant[20]	12.09	8.11		2-bit	KVQuant[20]	13.65	8.86
		KIVI[19]	10.64	7.39			KIVI[19]	10.59	8.25
		SKVQ[30]	10.61	7.07			SKVQ[30]	10.76	8.09
		SubKV	**10.41**	**6.97**			SubKV	**10.17**	**7.45**

4.2.2 | Perplexity results

Table 4 presents the results for the H2O-Danube3-500M, MobileLLM-600M, MobileLLM-1B, and TinyLlama-1.1B-chat models on the Wikitext2 and C4 datasets. The baseline configurations utilized by KVQuant[20], KIVI[19] and SKVQ[30] are included for reference. Table 4 results indicate that SubKV consistently outperforms baseline approaches, particularly in the context of 4-bit and 2-bit quantization. Specifically, When employing the SubKV method to quantize the KV Cache of H2O-Danube3-500M with 4-bit precision, the perplexity on the Wikitext2 and C4 datasets degrades by only 0.01 and 0.02, respectively. Furthermore, when quantizing the KV Cache to 2 bits, the models utilizing SubKV exhibit a perplexity degradation of merely 0.59 and 0.37, significantly lower than the degradation observed with other baseline methods. This trend is also evident in the other models. When quantizing the KV Cache of TinyLlama to 4 bits using the SubKV method, the perplexity degrades by only 0.03 and 0.02 on the Wikitext2 and C4 datasets, respectively. At 2-bit precision, the degradation is limited to 0.69 and 1.06, once again substantially lower than that of other baseline methods. Notably, SubKV incurs less accuracy loss when quantizing smaller models.

4.2.3 | Long Context evaluation and Memory analysis

To assess the capability of handling extended context windows, we perform perplexity evaluations on the Wikitext2 benchmark with progressively increasing input lengths, as demonstrated in Figure 7. The experimental results reveal that our quantization method achieves comparable performance to the full-precision baseline while significantly reducing memory consumption. For instance, when quantizing the KV Cache of the H2O-Danube3-500M model to 4 bits using SubKV, the model's perplexity remains nearly indistinguishable from that of the original model for both 4K and 8K input lengths. This minor trade-off enables SubKV to achieve a reduction in the model's KV Cache size by a factor of 3.31. Furthermore, when the KV Cache is quantized to 2 bits, the model demonstrates only minimal degradation in perplexity. Despite this slight decline in performance, SubKV successfully reduces the model's KV Cache size by as much as 5 times. These findings underscore the efficacy of SubKV in optimizing KV Cache utilization while preserving robust model performance.

4.2.4 | Efficiency comparison

To evaluate the wall-clock time efficiency of SubKV, following vLLM[61], we synthesize workloads based on ShareGPT[62], which contain input and output texts of real LLM services. On average, the dataset has an input prompt length of 161 and an output length of 338[61]. We increased the batch size until out of memory and reported the peak memory usage and throughput between SubKV (with residual length 32) and FP16 baseline for the H2O-Danube3-500M model. The hardware here is a single NVIDIA A800 GPU (80GB). As shown in Figure 8, with similar maximum memory usage, SubKV enables up to 1.6× larger batch size and gives 1.39× $\sim$ 2.17× larger throughput. This throughput number can grow larger with longer context length and output length.

4.3 | The Effect of Attention Sink-Aware Quantization

To evaluate the performance gains attributable to Attention Sink-Aware Quantization, we conducted a perplexity assessment on the Wikitext2 and C4 datasets, utilizing 4k input contexts. As illustrated in Table 5, Attention Sink-Aware Quantization consistently demonstrates significant performance improvements, particularly when employing lower bit-width quantization strategies. For instance, when quantizing the model's KV Cache to 4 bits, the benefits of Attention Sink-Aware Quantization are minimal, resulting in a perplexity reduction of only 0.03 at the most. In contrast, when the KV Cache is quantized to 2 bits, the gains become pronounced, with a substantial perplexity reduction of 1.32 observed specifically on the Wikitext2 dataset. It highlights the considerable performance enhancements facilitated by Attention Sink-Aware Quantization, particularly for extremely low-bit width quantization scenarios.

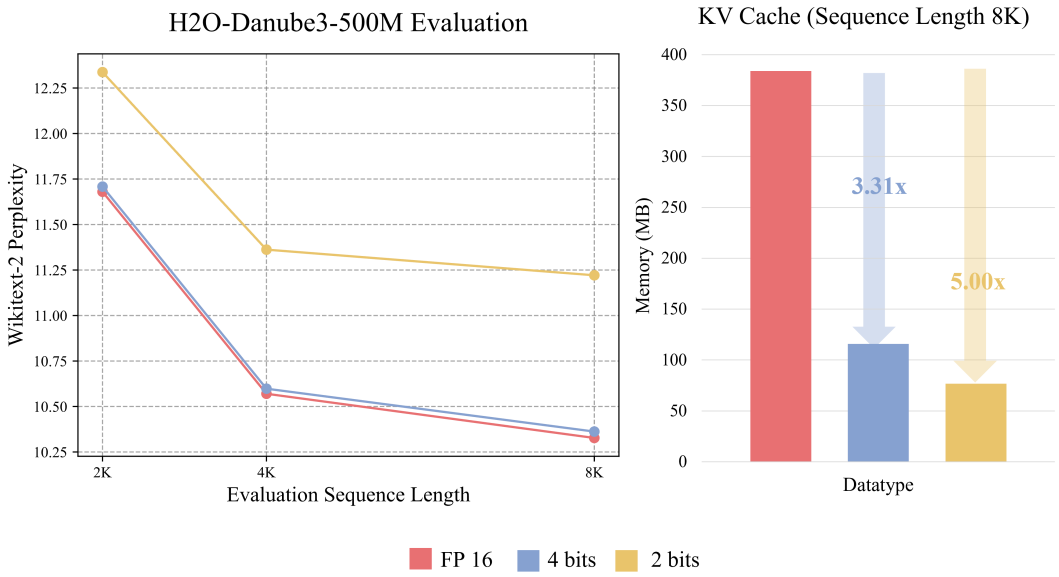

FIGURE 7 Perplexity results for the H2O-Danube3-500M model on the Wikitext2 Dataset. This figure illustrates the perplexity scores obtained by the H2O-Danube3-500M model when evaluated on the Wikitext2 dataset across various sequence lengths. A lower perplexity score indicates better predictive accuracy, allowing for a comprehensive assessment of the model's effectiveness in generating coherent and contextually relevant text.

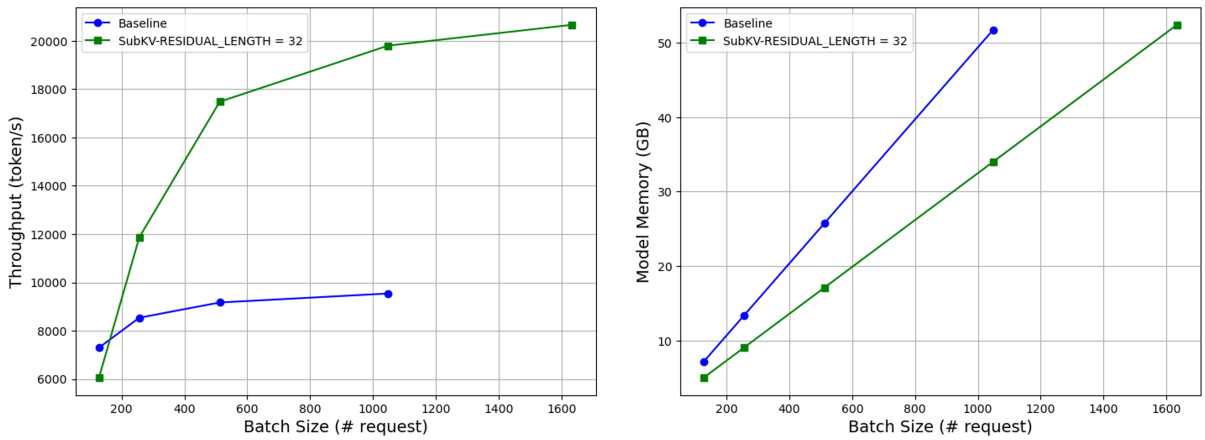

FIGURE 8 Memory usage and throughput comparison between 2-bit SubKV and FP16 baseline. SubKV can achieve higher throughput by enabling a larger batch size. We use H2O-Danube3-500M model as base model.

TABLE 5 Performance comparison of models with and without Attention Sink-Aware Quantization on Wikitext2 and C4 Datasets. This table presents results for models using 4k tokens input contexts.

Model		Method	Perplexity ↓	
			WikiText2	C4
H2O-Danube3-500M	FP16	-	10.57	13.90
	4-bit	SubKV-no-sink-att	10.61	13.93
		SubKV	**10.58**	**13.92**
	2-bit	SubKV-no-sink-att	12.54	14.98
		SubKV	**11.22**	**14.28**

4.4 | Ablation study

In this section, we conducted an ablation study of SubKV on the WikiText2 dataset[28] to systematically investigate the impact of group size G and window length N. The context length is set to 4k tokens, allowing us to assess how variations in these parameters influence the effectiveness of the model in processing extended text.

4.4.1 | The Effect of Group Size

We maintained a fixed window length of 96 while varying the group sizes to 8, 16, and 32. As shown in Table 6, the results demonstrate that group sizes of 8 and 32 yield comparable performance outcomes. It is important to emphasize that the zero-point and scaling factor, as discussed in Section 3.1, are calculated based on the selected group size. Therefore, the choice of group size significantly influences the effectiveness of KV Cache compression, particularly in the context of processing long input sequences.

4.4.2 | The Effect of Window Length

Keeping the group size fixed at 16, we investigated various window lengths of 16, 32, 64, and 128. The results presented in Table 7 reveal no consistent correlation between window lengths and model accuracy. Specifically, a window length of 128 yields favorable results, a length of 64 is associated with the poorest performance. We emphasize that, despite observing no significant differences among the window lengths of 32, 64, and 128, it is crucial to maintain a reasonably large window length, as it contributes to considerable performance enhancements, as demonstrated in Table 7.

TABLE 6 Ablation study of SubKV with varying Group Sizes (G). The window length is kept constant at 96, while the group sizes are varied at 8, 16, and 32.

Model	Group Size	PPL ↓
H2O-Danube3-500M	8	14.11
	16	10.87
	32	11.03

TABLE 7 Ablation study of SubKV with varying Window Length (N). The roup size is kept constant at 16, while the window length are varied at 16, 32, 64 and 128.

Model	Window Length	PPL ↓
H2O-Danube3-500M	16	10.96
	32	10.88
	64	11.01
	128	10.77

Original Model Output

Clarissa Dickson Wright, a former TV chef and co-star of the popular BBC Two show Two Fat Ladies, has passed away at the age of 66. She was known for her eccentricity, intelligence, and irreverence, which made her a beloved figure in the culinary world. Dickson Wright was the co-creator of the show alongside her partner, Jennifer Paterson. She was also a champion of hunting and shooting, and her passion for the outdoors led her to become a vegetarian.
Dickson Wright's career was marked by her involvement in various TV shows, including Breakfast, Lunch, and Dinner, where she and Paterson often traveled the country together in their motorcycle and sidecar. She was known for her irreverence and humor, which often led to her being labeled as "Krakatoa" by her critics.
In addition to her TV work, Dickson Wright was a prolific writer, publishing a book in 1998, "A History of English Food," which she wrote with her former partner. She also wrote a memoir, "Spilling the Beans," which detailed her life and experiences.
Dickson Wright's life was marked by her alcoholism and her battle with mental health issues. She was diagnosed with cancer in 1999 and passed away at the Royal Infirmary in Edinburgh. Her death was met with sadness and admiration from her fans and colleagues.
Dickson Wright's legacy continues to be celebrated, and her life and work have been the subject of numerous tributes and remembrances. Her contributions to the culinary world and her ability to bring joy and laughter to audiences have left a lasting impact.

SubKV-2bits Output

Clarissa Dickson Wright, a former Two Fat Ladies co-star and co-chef, passed away at the age of 66. She was known for her eccentricity, intelligence, and love for hunting and shooting. Dickson Wright was a brilliant cook, an erudite food historian, and a champion of hunting and shooting. She was a force of nature and a commanding presence on television, often seen in a motorcycle-sidecar with her partner, Jennifer Paterson. She was a champion of the countryside pursuits and a supporter of the Countryside Alliance. Dickson Wright was a woman of privilege, having a well-to-do upbringing and a career as a barrister. She was a former country girl who became a barrister and later a television star. She was a devoted wife and mother, and she was well-known for her love of champagne and her game. She was a woman who lived her life to the fullest and was celebrated for her intelligence and eccentricity.

Reference Answer

"Both of TV's "Fat Ladies" have sung. Clarissa Dickson Wright, one half of the BBC's "Two Fat Ladies" cooking duo, died in Edinburgh Saturday at age 66, the BBC reports. Wright was a former lawyer who filmed four of the "Fat Ladies" series, going on food-related road trips across the UK in a motorbike and sidecar with Jennifer Paterson, before Paterson died in 1999 from cancer. The New York Times describes Wright as a "rebel," both hosts as "irreverent and eccentric," and the recipes as "sometimes confounding." Wright's eclectic working life also included stints as a cook, an author, and a cookbook shop manager; she also ran a catering business, was a guild butcher, and once worked on a yacht in the Caribbean. In fact, she recently said, "I've had a fantastic life and I've done everything I could have wanted to do and more." It wasn't until her 40s, after she'd recovered from alcoholism, that she got into cooking seriously. As for the perhaps-controversial title of the show that brought her fame? "If you're fat you're fat," she once said. "I hate this modern-day political correctness, that you don't call things by their proper name." Her agent remembers Wright similarly in a statement: "Loved dearly by her friends and many fans all over the world, Clarissa was utterly non-PC and fought for what she believed in, always, with no thought to her own personal cost." There's no word on Wright's cause of death, but the Guardian reports that she had been undergoing treatment at a hospital since the beginning of the year.

FIGURE 9 Visualization of one generation example with H2O-Danube3-500M. Results are compared between the baseline model with full Cache, our SubKV-2bit, and the reference answer.

4.5 | Case Study

To visually demonstrate that models utilizing SubKV quantization retain their long-context response capabilities, we selected a case study from the MultiNews dataset within LongBench. MultiNews[53] involves a multi-document summarization task, requiring the model to generate summaries from multiple news articles, with a context length of 3,000 tokens for our example. As shown in Figure 9, despite SubKV compressing the model's KV Cache to an extremely low bit-width of 2-bit, the model continues to effectively respond to questions. Although the generated summary length is reduced compared to that of the original model, it still captures key points from the news articles. Specifically, the shorter summary maintains essential details, including the main events, key figures, and the overall implications of the stories. Critical aspects, such as the outcomes of political decisions, economic forecasts, and public reactions, are succinctly summarized, providing readers with a comprehensive understanding without unnecessary verbosity. This illustrates that even with a reduced summary length, critical insights are preserved, demonstrating the effectiveness of SubKV quantization in maintaining essential information. This case study clearly shows that SubKV can significantly compress the KV Cache while minimizing accuracy loss, allowing the model to retain performance even at a remarkably low precision of 2-bit.

5 | CONCLUSIONS AND FUTURE WORK

In this paper, we present SubKV, an accurate ultra-low precision quantization method specifically designed for KV cache management in sub-billion parameter LLMs. Recognizing the distinct distribution characteristics of keys and values in sub-billion parameter models, we employ Per-Channel Quantization for keys while implementing Per-Token Quantization for values. This dual approach allows us to effectively address the varying sensitivities and distributions of these components.

Moreover, we introduce a Dynamic Window Quantization strategy equipped with filtering rules. This innovative strategy significantly enhances the performance of KV Cache quantization for long context tasks by designating a small portion of the cache to retain full precision, thereby mitigating the adverse effects of quantization on critical data.

Additionally, we implement Attention Sink-Aware Quantization, which further improves the quantized model's performance by preserving full precision for the first token in the input sequence. By integrating these three strategies, we have achieved a quantization of the KV Cache to 2 bits with minimal precision loss, thereby maintaining the model's efficacy.

Looking ahead, our future work will focus on refining the dynamic window filtering rules and optimizing the kernel implementation to enhance the overall usability and performance of the SubKV quantization method. Through these advancements, we aim to further solidify the viability of ultra-low precision quantization in practical applications of large language models.

LIMITATIONS

Further experimentation is essential to comprehensively evaluate large language models (LLMs) as their use across diverse tasks grows. When quantizing LLMs to lower bit precision, it is critical to acknowledge that such transformations may disproportionately affect various model capabilities, underscoring the need for rigorous and holistic performance assessments of the quantized models. While our current evaluation encompasses multiple downstream tasks, including perplexity (PPL) and LongBench, we recognize that these metrics may not fully encapsulate the broader functional spectrum of LLMs. Furthermore, SubKV achieves substantial reductions in storage requirements for long context inference by compressing the KV Cache to extremely low-bit precision. However, these gains are accompanied by increased computational overhead. Although we have successfully implemented optimizations for GPU deployment, further improvement is required to address the challenges of optimizing quantization for edge devices, such as mobile platforms. Additionally, our latency benchmarking results currently focus on memory-bandwidth bound generation rather than prompt processing (where we need to compress multiple Keys and Values at once). Future work should aim to bridge these gaps to enable more efficient and generalized deployment.

ETHICS STATEMENT

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here. The development of quantization techniques for large language models (LLMs) holds the promise of democratizing access to these advanced systems by reducing deployment costs. This can empower a broader audience to benefit from sophisticated AI assistants, fostering innovation and enhancing productivity across various sectors. However, it is crucial to recognize that LLMs can inherit social biases embedded in their training data, including biases related to gender, race, and other demographic factors. These biases may manifest in the outputs of the models, potentially reinforcing harmful stereotypes and inequalities. While the quantization process improves model efficiency and resource utilization, it does not inherently address these underlying biases. Given this context, we urge caution in the utilization of quantized LLMs in real-world applications. Users and developers must remain vigilant about the implications of these biases and actively consider the ethical ramifications of deploying such technologies. Implementing additional safeguards and bias mitigation strategies is essential to ensure that the deployment of quantized LLMs aligns with ethical standards and promotes fairness in AI interactions. This commitment to responsible AI development is critical for fostering trust and ensuring that the benefits of these technologies are equitably distributed.

REFERENCES

1. Touvron H, Lavril T, Izacard G, et al. LLaMA: Open and Efficient Foundation Language Models. *ArXiv.* 2023;abs/2302.13971.
2. Touvron H, Martin L, Stone KR, et al. Llama 2: Open Foundation and Fine-Tuned Chat Models. *ArXiv.* 2023;abs/2307.09288.
3. Zhang S, Roller S, Goyal N, et al. OPT: Open Pre-trained Transformer Language Models. *ArXiv.* 2022;abs/2205.01068.
4. Scao TL, Fan A, Akiki C, et al. BLOOM: A 176B-Parameter Open-Access Multilingual Language Model. *ArXiv.* 2022;abs/2211.05100.
5. Wang Y, Kordi Y, Mishra S, et al. Self-Instruct: Aligning Language Models with Self-Generated Instructions. *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).* 2023:13484–13508.
6. Liu Z, Zhao C, Iandola F, et al. MobileLLM: Optimizing Sub-billion Parameter Language Models for On-Device Use Cases. *Proceedings of the 41st International Conference on Machine Learning.* 2024;235:32431–32454.
7. Pfeiffer P, Singer P, Babakhin Y, Fodor G, Dhankhar N, Ambati S. H2O-Danube3 Technical Report. *ArXiv.* 2024;abs/2407.09276.
8. Cui J, Ning M, Li Z, et al. Chatlaw: A Multi-Agent Collaborative Legal Assistant with Knowledge Graph Enhanced Mixture-of-Experts Large Language Model. *ArXiv.* 2023;abs/2306.16092.
9. L'ala J, O'Donoghue O, Shtedritski A, Cox S, Rodriques SG, White AD. PaperQA: Retrieval-Augmented Generative Agent for Scientific Research. *ArXiv.* 2023;abs/2312.07559.
10. Yuan Z, Niu L, Liu JW, et al. RPTQ: Reorder-based Post-training Quantization for Large Language Models. *ArXiv.* 2023;abs/2304.01089.
11. Zhu F, Lv Y, Chen Y, Wang X, Xiong G, Wang FY. Parallel Transportation Systems: Toward IoT-Enabled Smart Urban Traffic Control and Management. *IEEE Transactions on Intelligent Transportation Systems.* 2020;21(10):4063-4071.
12. Du J, Wu H, Xu M, Buyya R. Computation Energy Efficiency Maximization for NOMA-Based and Wireless-Powered Mobile Edge Computing With Backscatter Communication. *IEEE Transactions on Mobile Computing.* 2024;23(6):6954-6970.
13. Xue M, Wu H, Peng G, Wolter K. DDPQN: An Efficient DNN Offloading Strategy in Local-Edge-Cloud Collaborative Environments. *IEEE Transactions on Services Computing.* 2022;15(2):640-655.
14. Chen C, Li Y, Wang Q, Yang X, Wang X, Yang LT. An Intelligent Edge-Cloud Collaborative Framework for Communication Security in Distributed Cyber-Physical Systems. *IEEE Network.* 2024;38(1):172-179.
15. Wang X, Ren L, Yuan R, Yang LT, Deen MJ. QTT-DLSTM: A Cloud-Edge-Aided Distributed LSTM for CyberPhysicalSocial Big Data. *IEEE Transactions on Neural Networks and Learning Systems.* 2023;34(10):7286-7298.
16. Zhao WX, Zhou K, Li J, et al. A Survey of Large Language Models. *ArXiv.* 2023;abs/2303.18223.
17. Liu Z, Desai A, Liao F, et al. Scissorhands: Exploiting the Persistence of Importance Hypothesis for LLM KV Cache Compression at Test Time. *Advances in Neural Information Processing Systems.* 2023;36:52342–52364.
18. Xiao G, Lin J, Seznec M, Wu H, Demouth J, Han S. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. *Proceedings of the 40th International Conference on Machine Learning.* 2023;202:38087–38099.
19. Liu Z, Yuan J, Jin H, et al. KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache. *Proceedings of the 41st International Conference on Machine Learning.* 2024;235:32332–32344.

20. Hooper C, Kim S, Mohammadzadeh H, et al. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. *Advances in Neural Information Processing Systems.* 2024;37:1270–1303.
21. Yue Y, Yuan Z, Duanmu H, Zhou S, Wu J, Nie L. WKVQuant: Quantizing Weight and Key/Value Cache for Large Language Models Gains More. *ArXiv.* 2024;abs/2402.12065.
22. Dettmers T, Lewis M, Belkada Y, Zettlemoyer L. LLM.int8(): 8-bit matrix multiplication for transformers at scale. *Proceedings of the 36th International Conference on Neural Information Processing Systems.* 2024:30318 - 30332.
23. Kovaleva O, Romanov A, Rogers A, Rumshisky A. Revealing the Dark Secrets of BERT. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP).* 2019:4365–4374.
24. Beltagy I, Peters ME, Cohan A. Longformer: The Long-Document Transformer. *ArXiv.* 2020;abs/2004.05150.
25. Ge S, Zhang Y, Liu L, Zhang M, Han J, Gao J. Model Tells You What to Discard: Adaptive KV Cache Compression for LLMs. *The Twelfth International Conference on Learning Representations.* 2024.
26. Xiao G, Tian Y, Chen B, Han S, Lewis M. Efficient Streaming Language Models with Attention Sinks. *The Twelfth International Conference on Learning Representations.* 2024.
27. Zhang P, Zeng G, Wang T, Lu W. TinyLlama: An Open-Source Small Language Model. *ArXiv.* 2024;abs/2401.02385.
28. Merity S, Xiong C, Bradbury J, Socher R. Pointer Sentinel Mixture Models. *ArXiv.* 2016;abs/1609.07843.
29. Raffel C, Shazeer NM, Roberts A, et al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.*. 2019;21:140:1-140:67.
30. Duanmu H, Yuan Z, Li X, Duan J, ZHANG X, Lin D. SKVQ: Sliding-window Key and Value Cache Quantization for Large Language Models. *First Conference on Language Modeling.* 2024.
31. Ainslie J, Lee-Thorp J, Jong dM, Zemlyanskiy Y, Lebron F, Sanghai S. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing.* 2023:4895–4901.
32. Dettmers T, Svirschevski RA, Egiazarian V, et al. SpQR: A Sparse-Quantized Representation for Near-Lossless LLM Weight Compression. *The Twelfth International Conference on Learning Representations.* 2024.
33. Frantar E, Ashkboos S, Hoefler T, Alistarh D. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. *ArXiv.* 2022;abs/2210.17323.
34. Lin J, Tang J, Tang H, et al. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. *Proceedings of the Seventh Annual Conference on Machine Learning and Systems.* 2024;6:87–100.
35. Kim S, Hooper CRC, Gholami A, et al. SqueezeLLM: Dense-and-Sparse Quantization. *Proceedings of the 41st International Conference on Machine Learning.* 2024;235:23901–23923.
36. Shao W, Chen M, Zhang Z, et al. OmniQuant: Omnidirectionally Calibrated Quantization for Large Language Models. *The Twelfth International Conference on Learning Representations.* 2024.
37. Xiao G, Lin J, Seznec M, Wu H, Demouth J, Han S. SmoothQuant: accurate and efficient post-training quantization for large language models. *Proceedings of the 40th International Conference on Machine Learning.* 2023;202:38087 - 38099.
38. Zhao Y, Lin C, Zhu K, et al. Atom: Low-Bit Quantization for Efficient and Accurate LLM Serving. *Proceedings of the Seventh Annual Conference on Machine Learning and Systems.* 2024;6:196–209.
39. Zhang Z, Sheng Y, Zhou T, et al. H_2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. *Advances in Neural Information Processing Systems.* 2023;36:34661–34710.
40. Zeng Z, Hong Y, Dai H, Zhuang H, Chen C. ConsistentEE: A Consistent and Hardness-Guided Early Exiting Method for Accelerating Language Models Inference. *Proceedings of the AAAI Conference on Artificial Intelligence.* 2024;38(17):19506-19514.
41. Ge S, Zhang Y, Liu L, Zhang M, Han J, Gao J. Model Tells You What to Discard: Adaptive KV Cache Compression for LLMs. *Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@NeurIPS 2023).* 2023.
42. Liu Z, Desai A, Liao F, et al. Scissorhands: exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. *Proceedings of the 37th International Conference on Neural Information Processing Systems.* 2024;36:5234252364.
43. Ribar L, Chelombiev I, Hudlass-Galley L, Blake C, Luschi C, Orr D. SparQ Attention: Bandwidth-Efficient LLM Inference. *Proceedings of the 41st International Conference on Machine Learning.* 2024;235:42558–42583.
44. Li C, Peng Y, Liu G, Li Y, Yang X, Chen C. Efficient Vision Transformer for Human-Centric AIoT Applications Through Token Tracking Assignment. *IEEE Transactions on Consumer Electronics.* 2024;70(1):1029-1039.
45. Sheng Y, Zheng L, Yuan B, et al. FlexGen: high-throughput generative inference of large language models with a single GPU. *Proceedings of the 40th International Conference on Machine Learning.* 2023;202:31094 - 31116.
46. Kwon W, Li Z, Zhuang S, et al. Efficient Memory Management for Large Language Model Serving with PagedAttention. *Proceedings of the 29th Symposium on Operating Systems Principles.* 2023:611626.
47. Jin Y, Wu CF, Brooks D, Wei GY. S3: increasing GPU utilization during generative inference for higher throughput. *Proceedings of the 37th International Conference on Neural Information Processing Systems.* 2024;36:18015–18027.
48. Liu R, Bai H, Lin H, et al. IntactKV: Improving Large Language Model Quantization by Keeping Pivot Tokens Intact. *Findings of the Association for Computational Linguistics ACL 2024.* 2024:7716–7741.
49. Jiang AQ, Sablayrolles A, Mensch A, et al. Mistral 7B. *ArXiv.* 2023;abs/2310.06825.
50. Dauphin YN, Fan A, Auli M, Grangier D. Language Modeling with Gated Convolutional Networks. *Proceedings of the 34th International Conference on Machine Learning.* 2017;70:933–941.
51. Dasigi P, Lo K, Beltagy I, Cohan A, Smith NA, Gardner M. A Dataset of Information-Seeking Questions and Answers Anchored in Research Papers. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* 2021:4599–4610.
52. Zhong M, Yin D, Yu T, et al. QMSum: A New Benchmark for Query-based Multi-domain Meeting Summarization. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* 2021:5905–5921.
53. Fabbri A, Li I, She T, Li S, Radev D. Multi-News: A Large-Scale Multi-Document Summarization Dataset and Abstractive Hierarchical Model. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics.* 2019:1074–1084.
54. Li X, Roth D. Learning Question Classifiers. *COLING 2002: The 19th International Conference on Computational Linguistics.* 2002.

55. Hovy E, Gerber L, Hermjakob U, Lin CY, Ravichandran D. Toward Semantics-Based Answer Pinpointing. *Proceedings of the First International Conference on Human Language Technology Research.* 2001.
56. Joshi M, Choi E, Weld D, Zettlemoyer L. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).* 2017:1601–1611.
57. Gliwa B, Mochol I, Biesek M, Wawer A. SAMSum Corpus: A Human-annotated Dialogue Dataset for Abstractive Summarization. *Proceedings of the 2nd Workshop on New Frontiers in Summarization.* 2019:70–79.
58. Guo D, Xu C, Duan N, Yin J, McAuley J. LongCoder: a long-range pre-trained language model for code completion. *Proceedings of the 40th International Conference on Machine Learning.* 2023;202:1209812107.
59. Liu T, Xu C, McAuley J. RepoBench: Benchmarking Repository-Level Code Auto-Completion Systems. *The Twelfth International Conference on Learning Representations.* 2024.
60. Bai Y, Lv X, Zhang J, et al. LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding. *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).* 2024:3119–3137.
61. Kwon W, Li Z, Zhuang S, et al. Efficient Memory Management for Large Language Model Serving with PagedAttention. *Proceedings of the 29th Symposium on Operating Systems Principles.* 2023:611626.
62. ShareGPT Team. https://sharegpt.com/; 2023.