Prefixing Attention Sinks can Mitigate Activation Outliers for Large Language Model Quantization

Seungwoo Son[1,2*], **Wonpyo Park**[2], **Woohyun Han**[2], **Kyuyeun Kim**[2], **Jaeho Lee**[1,2†]
[1]POSTECH [2]Google
{swson, jaeho.lee}@postech.ac.kr
{wppark, woohyun, kyuyeunk}@google.com

Abstract

Despite recent advances in LLM quantization, activation quantization remains to be challenging due to the activation outliers. Conventional remedies, *e.g.*, mixing precisions for different channels, introduce extra overhead and reduce the speedup. In this work, we develop a simple yet effective strategy to facilitate per-tensor activation quantization by preventing the generation of problematic tokens. Precisely, we propose a method to find a set of key-value cache, coined *CushionCache*, which mitigates outliers in subsequent tokens when inserted as a prefix. CushionCache works in two steps: First, we greedily search for a prompt token sequence that minimizes the maximum activation values in subsequent tokens. Then, we further tune the token cache to regularize the activations of subsequent tokens to be more quantization-friendly. The proposed method successfully addresses activation outliers of LLMs, providing a substantial performance boost for per-tensor activation quantization methods. We thoroughly evaluate our method over a wide range of models and benchmarks and find that it significantly surpasses the established baseline of per-tensor W8A8 quantization and can be seamlessly integrated with the recent activation quantization method.

1 Introduction

Tremendous capabilities of large language models (LLMs) come with a tremendous computational cost. Modern language models often have over hundreds of billions of parameters, requiring significant memory and computation for prediction and training. For instance, OPT-175B (Zhang et al., 2022), one of the most popular open-sourced language models, requires at least 350GB of memory and the order of 10^{18} floating point operations to generate a new token[1] (Hoffmann et al., 2022).

Quantization is an effective strategy to reduce the computational cost of LLMs. Recent works demonstrate that the precision of LLM weight parameters can be greatly reduced by post-training quantization (PTQ), with minimal degradations in its generation quality. For example, Huang et al. (2024) shows that one can quantize the weights of the LLaMA3-70B to 4 bits, with less than 0.5%p drop in its zero-shot prediction accuracy. Roughly, the reduced precision translates into 4× increase in the generation throughput, and similar reduction in memory requirements (Lin et al., 2024).

LLM activations, however, remain challenging to be quantized. The key obstacle is the activation outlier, *i.e.,* a small number of activations that are substantially larger than others (Bondarenko et al., 2021; Dettmers et al., 2022; Sun et al., 2024). Such outliers elongate the quantization range and flattens out most non-outlier activations, leading to large performance losses even at W8A8 quantization.

To address this issue, recent works propose to mitigate outliers based on various relaxations of the stringent *static, per-tensor* quantization. One line of work applies quantization separately to each channel depending on the outlier proneness (Bondarenko et al., 2021; Dettmers et al., 2022). These methods, however, are difficult to be implemented on conventional hardwares. Another line of work reparameterizes the activations and weights in a way that the impact of outliers are amortized (Xiao et al., 2023; Ashkboos et al., 2024). These algorithms focuses on attaining high generative quality by adopting per-token or dynamic quantization, leaving the most hardware-friendly option—static per-tensor quantization—less explored.

To fill this gap, we take a novel approach for mitigating activation outliers in LLMs. In particular, we focus on answering the following key question:

Can we find a good ***prefix*** *that mitigates the activation outliers in the subsequent*

[*] Work done during internship at Google.
[†] Contact: `jaeho.lee@postech.ac.kr`
[1] assuming the context length 2048

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 2242–2252
November 12-16, 2024 ©2024 Association for Computational Linguistics

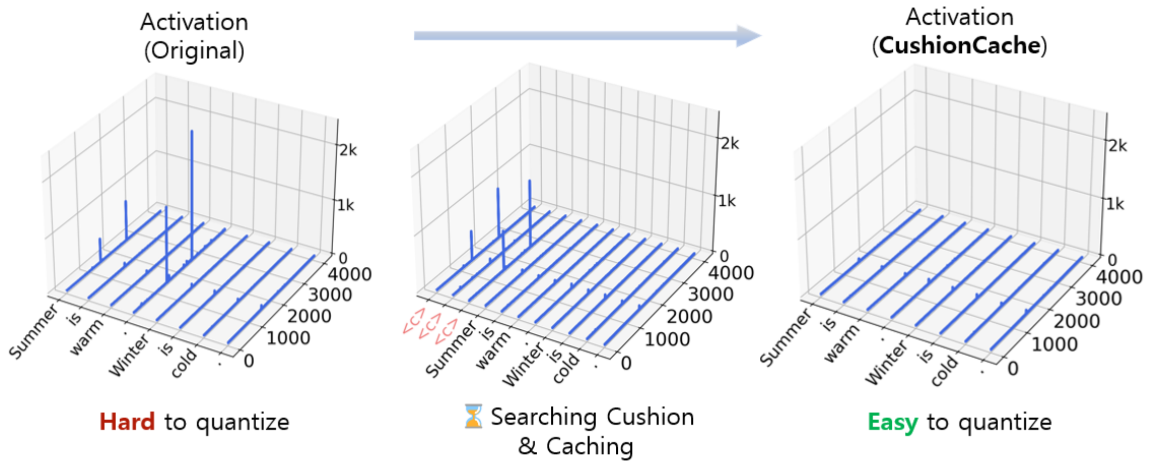

Figure 1: **Activation magnitudes in LLaMA2-7B, before and after CushionCache.** CushionCache mitigates the activation outliers in LLMs by inserting and tuning the several prefix tokens to the model, which acts as an attention sink. Adding such sink tokens alleviates outliers in the subsequent tokens and enables a better activation quantization of the model with coarse quantization granularities.

tokens on a pretrained LLM?

Our answer is positive; we develop a very simple yet effective method, coined ***CushionCache***, to discover a prefix[2] which reduces the outlier in the following tokens processed by the given LLM. By inserting this prefix, one then can quantize the activations of the LLM with much smaller quantization error, leading to an improved generation quality.

To design our method, we draw inspirations from a recent observation that the outliers may originate from *attention sinks* (Bondarenko et al., 2023)—the "no-operation" tokens that receive much attention from other tokens (Xiao et al., 2024). By adding sink-like tokens as a prefix, one may be able to separate out outlier activations as well, rendering the subsequent tokens outlier-free. In a nutshell, our method works in two steps.

1. *Greedy initialization.* We search for a sequence of sink-like prompt tokens in a greedy manner, so that the activations of the subsequent tokens are less prone to outliers (Section 4.1).
2. *Quantization-aware prefix tuning.* We train the greedily initialized prefix further to minimize the combined loss of the prediction loss and quantization error (Section 4.2).

Our experiments demonstrate that the proposed CushionCache is highly effective in making LLMs more quantizable. The technique is versatile, consistently improving the quantized performance of LLMs under various scenarios, from per-token to per-tensor static quantization. The method can also be seamlessly combined with existing quantization algorithms to further boost their performances. To summarize, we contribute the following.

- We introduce CushionCache, a new prefix discovery method for mitigating LLM outliers to improve the quantization performance.
- Through extensive experiments, we show that CushionCache can consistently improve the performance of quantized LLMs under a wide range of setup. In particular, we improve the prior state-of-the-art W8A8 per-tensor static range quantization of LLaMA3-8B over 30%p in zero-shot accuracy on downstream tasks.
- Through our analysis, we demonstrate that CushionCache effectively replaces the role of attention sink tokens.

2 Related Work

Outliers in LLMs. The fact that there exists usually large entries in LLM activations, or outliers, has been reported by multiple works. Kovaleva et al. (2021) and Timkey and van Schijndel (2021) report the existence of outliers in large transformer-based language models (*e.g.*, BERT), and find that they appear mostly in a small number of channels and layers. Bondarenko et al. (2021) make a similar observation in the context of quantization, and finds that quantizing the activations lead to a large degradation in generation quality; the work also reports that semantically meaningless tokens can have higher tendencies to have outliers.

[2] more precisely, the key-value cache; we only care about the keys and values, rather than the token itself,

Dettmers et al. (2022) confirm the same finding while quantizing GPT-scale models, and studies how the model scale affects the prevalence of outliers over tokens and layers. More recently, Sun et al. (2024) investigates a similar phenomenon in newer LLM variants and confirms that certain tokens are more likely to suffer from outliers.

Per-channel activation quantization. A line of work proposes to mitigate outliers in LLM activation quantization by applying different scaling factors or precision to each channel. Bondarenko et al. (2021) splits activation channels into several groups and perform quantization on each group. LLM.int8() (Dettmers et al., 2022) applies higher precision (*e.g.*, FP16) to a small number of outlier-prone channels, while quantizing the other channels to lower bits (*e.g.*, INT8). These works, however, are difficult to be implemented in conventional hardwares, as they requires scaling along the contracting dimension of matrix multiplication.

Per-token, with reparameterization. Another line of work proposes to quantize the activations per-token to reduce the impact of outliers with better hardware acceleration. Many of these works adopt *reparameterization* of weights to mitigate the outliers further. ZeroQuant (Yao et al., 2022) applies per-tensor quantization and knowledge distillation to achieve reasonable INT8 quantization performance. SmoothQuant (Xiao et al., 2023), Outlier Suppression+ (Wei et al., 2023), and OmniQuant (Shao et al., 2024) migrates the activation magnitudes to the weights to normalize the scales of the activations. More recently, QuaRot (Ashkboos et al., 2024) rotates the activations so that the outlier magnitudes are distributed over multiple axes in the reparametrized space. While these methods are effective, per-token quantization are typically slower than per-tensor quantization at the same quantization precision as it requires larger scale which has a size of the number of tokens.

Per-tensor quantization. Notably, Xiao et al. (2023) also provides two options for per-tensor activation quantization: one with dynamic quantization range, and another with static range. While these options tend to be faster than per-token (with static range being the fastest), their generation quality is much lower than per-token, especially on recent models such as LLaMA3 (Touvron et al., 2023).

Attention sinks and outliers. Recent works report an intriguing phenomenon in large transformers, termed *attention sink*. Xiao et al. (2024) find that a small number of semantically meaningless tokens, usually at the beginning of the sequence, tend to receive unusually large attention. Darcet et al. (2024) make a similar observation for vision transformers, and show that training ViTs with additional meaningless tokens can help make the attention structures more semantically meaningful. Bondarenko et al. (2023) hypothesize that the sink tokens may be the root cause of the activation outliers, and propose a new architecture that prevents the outliers from emerging when pretrained from scratch. Our work shares a similar intuition, but critically differs in that we mitigate outliers by fine-tuning the pretrained LLM. This means no modification to the network architecture is needed and does not need to train the model from scratch.

3 Preliminaries

Key-value cache. Modern language models, typically based on decoder-only architecture, are built as a sequence of transformer blocks which process a sequence of tokens to predict the next token (Vaswani et al., 2017). That is, at each decoding step, the transformer $f(\cdot)$ performs:

$$t_{n+1} = f(t_1, t_2, \dots, t_n) \tag{1}$$

where $t_1, \cdots, t_n$ are the preceding tokens used as context. LLM iteratively applies Eq. (1) autoregressively to generate text as a sequence of tokens.

As the context length grows, the computational cost to process all previous tokens also grows larger, slowing down the generation significantly. A popular solution is to cache and reuse the keys and values of the preceding tokens computed during the previous iteration. This trick relies on the fact that preceding tokens affect the outcome of the current token only through their keys and values:

$$(s_{n,1}, \dots, s_{n,n}) = \text{Attention}(q_n, k_{1:n})$$
$$o_n = \sum_{i=1}^{n} v_i \cdot s_{n,i}$$

where q_i, k_i, v_i denotes the query, key, and value vectors for each token and $s_{i,j}$ denotes the attention score for the i-th token query on j-th token key. By storing and reusing the cached values (called KV cache), one only needs to process new tokens as:

$$t_{n+l+1} = f(t_{n+1:n+l} \mid k_{1:n}, v_{1:n}), \tag{2}$$

where $t_{n+1:n+l}$ denotes l tokens given at the current step and $k_{1:n}, v_{1:n}$ are the keys and values of the preceding context of length n, computed during the previous iteration. During the prefill phase, l may be the length of the prompt, and during the decoding phase, we can simply use $l = 1$, processing only a single token at a time.

Quantization. Quantization is an act of casting a high-precision tensor (typically FP) into a lower-precision tensor (typically INT), to save the memory to store and computation to process the tensor. In neural network quantization, a popular choice is the *linear quantization*, which performs

$$\mathbf{X}_{\text{int}} = \text{round}((\mathbf{X}_{\text{fp}} - z)/s), \quad (3)$$

where $\mathbf{X}_{\text{int}}, \mathbf{X}_{\text{fp}}$ denotes the quantized and original tensors, $z, s \in \mathbb{R}$ denote zero-point and scaling factor, and $\text{round}(\cdot)$ denotes the rounding operation. The scaling factor is typically selected as

$$s = \frac{\max(\mathbf{X}_{\text{fp}}) - \min(\mathbf{X}_{\text{fp}})}{2^{N-1} - 1}, \quad (4)$$

where N denotes the number of bits for the integer format. The zero-point is determined as either $z = \min(\mathbf{X}_{\text{fp}})$ or $z = 0$, for the asymmetric and symmetric quantization, respectively.

Activation quantization with static range. By quantizing both activation and weight matrices, one can avoid performing computation-heavy FP matrix multiplications. That is, for the case of symmetric quantization (for simplicity), we approximate:

$$\mathbf{W}_{\text{fp}}\mathbf{X}_{\text{fp}} \approx s_{\text{W}} s_{\text{X}} \cdot \mathbf{W}_{\text{int}}\mathbf{X}_{\text{int}}, \quad (5)$$

where the right-hand side can be computed using an integer matrix multiplication, and a single multiplication of FP16/32 quantities (for scaling factors). The combined scaling factor need not be multiplied back to the matrix immediately, and can be used in the subsequent operations directly.

In many cases, the scaling factors $s_{\text{W}}, s_{\text{X}}$ can be pre-computed based on the validation set statistics. This method, called *static-range quantization*, enables more acceleration than computing these values dynamically during the inference.

Outliers and complications. In LLMs, the activation $\mathbf{X}_{\text{fp}}$ tends to have a very large entry (Dettmers et al., 2022; Sun et al., 2024). In such case, the magnitude of $\max(\mathbf{X}_{\text{fp}})$ and $\min(\mathbf{X}_{\text{fp}})$ will be very large, making the scaling factor s_{X} very large. This leads to a high sparsity in the tensor $\mathbf{X}_{\text{int}}$, and a much degraded generation quality.

This problem can be alleviated in various ways: One can change the scaling factor dynamically over time (*i.e.*, per-tensor dynamic quantization), or apply different scaling factors for each channel or token (*i.e.*, per-channel/token quantization). As these methods require on-the-fly computations of scaling factors, the methods are typically slower.

Granularity and the communication cost. The drawback of finer quantization granularity becomes more significant in the distributed setup, as it affect the communication cost between nodes.

To see this, consider the case of multiplying matrices with tensor parallelism, *e.g.*, Megatron-LM (Shoeybi et al., 2019). Comparing with the per-tensor static quantization, per-tensor dynamic quantization requires an additional AllReduce operation over the nodes to aggregate the (high-precision) scaling factor. The overhead is even more significant for per-token dynamic quantization, as the number of scaling factors is multiplied by the number of tokens, increasing the cost of AllReduce.

4 Method

We now describe CushionCache, an algorithm to find a prefix which can mitigate activation outliers in the subsequent tokens, thereby alleviating the quality degradation from activation quantization.

CushionCache aims to find a set of prefix that minimizes the quantization error of the activations. More concretely, let $\mathbf{X}_i$ denote activation of a transformer block for the input token t_i. Our goal is to minimize the squared difference between the original and the quantized activations, *i.e.*,

$$L_{\text{q}}(t_1, ..., t_n) = \sum_{i=1}^{n} \|\mathbf{X}_i - q(\mathbf{X}_i)\|_2^2, \quad (6)$$

where $q(\cdot)$ denotes the quantization function, specified as $q(\mathbf{X}) = s \cdot \text{round}((\mathbf{X} - z)/s) + z$. In practice, we consider the summation of the error L_q of all transformer blocks, but we omit this for the notational simplicity. Similarly, we define $L_q(t_{1:n}|p_{1:m})$ as the sum of squared error for $t_{1:n}$ given the prefix $p_{1:n}$, where the scaling factor s and zero-point z are determined for $t_{1:n}$ only.

We hypothesize that there exist prefix tokens p that can reduce the expected activation quantization error of the tokens. That is, we find

$$\hat{p}_{1:m} = \arg\min_{p_{1:m}} \mathbb{E}\left[L_q(t_{1:n} \mid p_{1:m})\right], \quad (7)$$

where the expectation is taken over the probability distribution of tokens $t_{1:n}$. Once we find such prefix $\hat{p}_{1:m}$, their keys and values are cached and reused at the inference time to avoid redundant computation:

$$t_{n+1} = f(t_{1:n} \mid \hat{k}_{1:m},\ \hat{v}_{1:m}), \tag{8}$$

where $\hat{k}$ and $\hat{v}$ corresponds to the key-value caches of the prefix $\hat{p}_{1:n}$, which we call *CushionCache*.

We solve the minimization (eq. 7) with a strategy based on prefix tuning. This is done in two steps: Initializing prefixes based on greedily searched prompts (Section 4.1), and Quantization-aware Prefix tuning (Section 4.2).

4.1 Greedy Prefix Search

We carefully initialize the prefix as the prefix tuning is known to be very sensitive to initial values. We follow Li and Liang (2021) to search for the prefix that are activations of hard prompt tokens, *i.e.*, input tokens that correspond to real text. As the search complexity grows exponentially with respect to the embedding size, we propose to use a greedy search algorithm with tailored heuristics.

In a nutshell, our method is a greedy search with ***early stopping***. We add new tokens to the prompt one-by-one, selected to minimize the quantization error. If the new token does not decrease the error much, we stop adding to prevent overfitting and computational overhead from long prompts.

Concretely, at each step, we first draw a single sample text $t_{1:n}$ from the dataset; we use the C4 dataset (Raffel et al., 2020), which is commonly used for calibration or validation purposes, to draw a sentence of length $n = 512$. Then, based on the current state of prompts $p_{1:k}$, we search for the next prompt token p_{k+1} by solving

$$p_{k+1} = \arg\min_{p \in \mathcal{E}} L_q(t_{1:n} | p_{1:k}, p), \tag{9}$$

where $\mathcal{E}$ denotes the embedding table; we can solve this problem rapidly by *batched inference*. If the discovered new token reduces the quantization error by some fraction $\tau > 0$, *i.e.*, satisfies

$$L_q(t_{1:n} | p_{1:k+1}) < \tau \cdot L_q(t_{1:n} | p_{1:k}), \tag{10}$$

then we append this token to the prompt and proceed to the next iteration. Otherwise, or if the max length is met, we stop searching for a new token. We use $\tau = 0.5$ for all experiments, which consistently shows a good performance.

Algorithm 1 Greedy prefix search

```
Require: validation dataset $D$, embedding table $\mathcal{E}$, max length $m$, threshold $\tau$
$p = [\,]$                                          ▷ initialize the prompt
while len($p$) $< m$ do
    $t \sim \mathrm{Unif}(D)$                           ▷ draw a text
    $p^* = \arg\min_{p' \in \mathcal{E}} L_q(t|p, p')$.
    if $L_q(t|p, p^*) > \tau \cdot L_q(t|p)$ then
        break
    end if
    $p$.append($p^*$)                                ▷ add new token
end while
return $p$
```

Note that this algorithmic design provides some flexibility. More specifically, one can initialize the prompt with nonempty sequence before the search, as a heuristic that can help speed up the prompt search procedure. We find that filling in nonsemantic words, *e.g.*, `<bos>` or `\n`, is particularly useful; this observation is well-aligned with the findings of Bondarenko et al. (2021); Sun et al. (2024).

4.2 Quantization-aware Prefix Tuning

Using the intermediate activations of the greedily-searched prompt as an initial prefix, we fine-tune the CushionCache via prefix tuning (Li and Liang, 2021). Precisely, we freeze the model parameters and train the prefix with the loss

$$L = L_{\text{pred}} + \lambda \cdot L_q \tag{11}$$

where L_{pred} is the cross entropy loss for the next-token prediction and λ is a hyperparameter that balances two losses. Here, we apply stop-grad to scaling factors and zero-points of the quantization function, as is typical in quantization-aware training literature (Jacob et al., 2018).

By optimizing this loss function, we ensure that the CushionCache not only improves the prediction accuracy but also minimizes the quantization error. This tuning does not require excessive amount of memory, as we only train the prefix.

5 Experiments

5.1 Experimental Setup

Models. We evaluate our method on five LLM models: LLaMA2 and 3 (Touvron et al., 2023), Mistral (Jiang et al., 2023), OPT (Zhang et al., 2022) and BLOOM (Le Scao et al., 2022).

WikiText-2 (↓)	LLaMA2-7B	LLaMA3-8B	Mistral-7B-v0.1	OPT-6.7B	BLOOM-7B
FP16	5.47	6.13	5.25	10.86	11.37
Per-tensor Static	9250.33	9759.46	85.51	11.45	11.93
+ CushionCache (ours)	5.98 (-99.9%)	7.41 (-99.9%)	5.84 (-93.2%)	11.00 (-3.9%)	11.50 (-3.6%)
SmoothQuant-O3	15439.73	14022.91	618.27	10.85	11.55
+ CushionCache (ours)	5.87 (-99.9%)	7.37 (-99.9%)	5.60 (-99.1%)	10.68 (-1.6%)	11.38 (-1.5%)
Per-tensor Dynamic	8.01	23.86	67.86	11.73	11.81
+ CushionCache (ours)	5.69 (-29.0%)	7.30 (-69.4%)	5.59 (-91.8%)	10.99 (-6.3%)	11.43 (-3.2%)
SmoothQuant-O2	8.13	25.12	66.16	10.87	11.59
+ CushionCache (ours)	5.66 (-30.4%)	7.29 (-71.0%)	5.56 (-91.6%)	10.68 (-1.7%)	11.39 (-1.7%)
Per-token Dynamic	5.47	6.22	5.30	11.20	11.47
+ CushionCache (ours)	5.37 (-1.8%)	6.15 (-1.1%)	5.21 (-1.7%)	10.77 (-3.8%)	11.37 (-0.9%)
SmoothQuant-O1	5.49	6.19	5.27	10.86	11.38
+ CushionCache (ours)	5.36 (-2.4%)	6.15 (-0.6%)	5.20 (-29.0%)	10.67 (-1.7%)	11.35 (-0.3%)

Table 1: **Perplexity of W8A8-quantized LLMs on raw-WikiText2.** Green denotes the relative decrease.

7 Zero-shot Tasks (↑)	LLaMA2-7B	LLaMA3-8B	Mistral-7B-v0.1	OPT-6.7B	BLOOM-7B
FP16	65.63	68.83	69.14	60.50	56.20
Per-tensor Static	36.37	35.86	48.83	57.94	55.87
+ CushionCache (ours)	64.47 (+28.10)	67.85 (+31.99)	67.75 (+18.91)	59.85 (+1.91)	55.91 (+0.04)
SmoothQuant-O3	36.32	36.22	37.45	60.61	55.96
+ CushionCache (ours)	64.67 (+28.35)	66.99 (+30.77)	68.39 (+30.94)	60.87 (+0.26)	56.66 (+0.75)
Per-tensor Dynamic	61.94	58.94	52.02	59.23	56.46
+ CushionCache (ours)	65.34 (+3.40)	68.66 (+9.72)	69.02 (+17.00)	60.28 (+1.05)	58.47 (+2.01)
SmoothQuant-O2	61.24	58.67	51.08	60.57	56.14
+ CushionCache (ours)	65.65 (+4.41)	68.74 (+10.07)	69.15 (+18.07)	60.60 (+0.03)	58.99 (+2.85)
Per-token Dynamic	65.43	68.92	68.90	59.48	56.55
+ CushionCache (ours)	65.78 (+0.35)	68.58 (-0.34)	69.83 (+0.93)	60.65 (+1.17)	56.72 (+0.17)
SmoothQuant-O1	65.64	68.64	69.09	60.55	56.35
+ CushionCache (ours)	65.97 (+0.33)	68.78 (+0.14)	69.99 (+0.90)	61.01 (+0.46)	56.80 (+0.45)

Table 2: **Average zero-shot accuracies of W8A8-quantized LLMs.** We average over LAMBADA, HellaSwag, PIQA, WinoGrande, OpenBookQA, RTE, and COPA. Green is the accuracy gain and red is the drop.

Datasets. We measure the perplexity on the held-out set of WikiText-2 validation dataset (Merity et al., 2016). For zero-shot evaluation, we use seven tasks from the LM evaluation harness benchmark by EleutherAI (Gao et al., 2023). Precisely, we use LAMBADA, HellaSwag, PIQA, WinoGrande, OpenBookQA, RTE, and COPA datasets.

Base algorithms. We apply CushionCache on two base activation quantization algorithms: Naïve activation quantization and SmoothQuant (Xiao et al., 2024). We consider three different scenarios: Per-tensor static, per-tensor dynamic, and per-token dynamic quantization. Note that for each case, the SmoothQuant has a corresponding version, called O3, O2, and O1, respectively.

Configuration: Quantization. We mostly follow the setup of Li et al. (2024) and the TensorFlow default. We use symmetric group-wise quantization for model weights, and asymmetric quantization for the activations. For SmoothQuant, we use the migration strength $\alpha = 0.8$, which worked consistently well throughout our experiments. For static range quantization, we calibrate using the training split of WikiText-2 (Merity et al., 2016).

Configuration: Prefix tuning. We follow the setup of Li and Liang (2021) and tune for 2 epochs. We set the hyperparameter $\lambda = 0.01$.

5.2 Main Results: W8A8 Quantization

In Tables 1 and 2, we provide the performance achieved by the quantized language models, quantized with and without the proposed CushionCache. We report the WikiText perplexity and zero-shot accuracy in the tables, respectively.

For per-tensor static range quantization, CushionCache successfully improves the performance of the model; the boost is quite substantial in LLaMA and Mistral, often providing over 30%p gains in terms of zero-shot accuracies. Intriguingly, the gain is much more pronounced in LLaMA-style models, which adopt the pre-LayerNorm and gated linear units. For per-tensor dynamic range quantization, similarly, we make consistent improvements

LLaMA3-8B	Zero-shot acc. (%)
FP16	68.83
Per-tensor Dynamic	58.94
+ Greedy-searched init.	67.78 (+8.84)
+ Prefix tuning	68.13 (+0.35)
+ Quantization-aware loss	68.66 (+0.53)

Table 3: **Ablation study.** We compare the contribution of each algorithmic component by sequentially adding them. We apply W8A8 per-tensor dynamic quantization on the LLaMA3-8B model.

Per-Token Dyn.	Perf.	LLaMA3-8B	Mistral-7B
FP16	ppl (↓)	6.13	5.25
	acc.(↑)	68.83	69.14
SmoothQuant-O1	ppl	6.93	5.49
(W6A6)	acc.	66.72	67.51
+ CushionCache	ppl	6.74 (-2.7%)	5.40 (-1.6%)
	acc.	67.60 (+0.88)	68.42 (+0.91)
SmoothQuant-O1	ppl	130.32	18.57
(W4A4)	acc.	40.25	51.11
+ CushionCache	ppl	29.09 (-77.7%)	12.45 (-33.0%)
	acc.	48.78 (+8.53)	55.58 (+4.47)

Table 4: **W6A6/W4A4 quantization.** We additionally evaluate per-token quantization with lower bits, as W8A8 does not degrade much performance in general.

over both vanilla quantization and SmoothQuant.

For per-token dynamic quantization, the gain is somewhat marginal, as the base quantization algorithms already tend to achieve a close performance to the FP16 model; we revisit per-token case for lower precision in Section 5.4.

5.3 Ablation Study

In Table 3, we sequentially add our key algorithmic components to validate their efficacy. In particular, the components are (1) greedy-searched initial value, (2) prefix tuning, and (3) the quantization-error-based regularizer.

We observe that each component makes nontrivial contributions for achieving near-FP16 zero-shot accuracy. Interestingly, we find that the greedy-searched initialization is especially effective, contributing ∼91% of the accuracy gain. This suggests that our search mechanism can be used as a compute-light standalone method in the cases where it is difficult to conduct prefix-tuning, due to a limited on-device memory.

5.4 4/6-bit Per-token Quantization

To confirm the effectiveness of CushionCache under per-token dynamic quantization, we additionally evaluate with a lower precision (Table 4). In particular, we use W6A6 and W4A4.

Model	Top-1	Top 10%	Median
LLaMA2-7B	2461.40	0.59	0.23
+ CushionCache (ours)	25.83	0.59	0.24
LLaMA3-8B	288.32	0.16	0.06
+ CushionCache (ours)	4.94	0.16	0.06
Mistral-7B-v0.1	352.05	0.12	0.04
+ CushionCache (ours)	3.51	0.12	0.04

Table 5: **Top-1, top 10%, and the median activation magnitudes of three LLMs.** Here, we inspect the input activations to the last transformer block.

The results confirm that the proposed CushionCache is also effective in boosting the quantization performance of per-token activation quantization algorithms. In particular, CushionCache helps keeping the accuracy degradation quite low (∼1%p) for W6A6 quantization of both LLaMA3 and Mistral.

5.5 Other experiments

In the Appendix A, we provide additional experiments results on the following topics:

- Evaluations on MMLU dataset (Appendix A.1)
- Latency measurements (Appendix A.2)
- Compatibility with other quantization methods (Appendix A.3)

6 Analysis

We now conduct a brief sanity check. In particular, we ask the following questions.

- Did the outliers disappear? (Section 6.1)
- Did the CushionCache really replace the role of attention sink? (Section 6.2)
- Will it be computationally viable to run CushionCache for large models? (Section 6.3)

6.1 Change of Activation Magnitudes

In Table 5, we report various order statistics of the activation magnitudes that appear in LLaMA2/3 and Mistral. In particular, we focus on the input activations to the last transformer block of these models, and measure the top-1, top 10%, and median (*i.e.*, top 50%) activation magnitude. We average over ten samples, with a sequence length 4096.

The effect of CushionCache is quite dramatic. In particular, we find that the CushionCache can reduce the scale of the activation outlier to 1-2% of the previous value. The ratio between the top-1 and the median decreases from roughly 10,000:1 to 100:1. We also note that the other order statistics, *i.e.*, top 10% and median, remains roughly the same before and after the CushionCache.

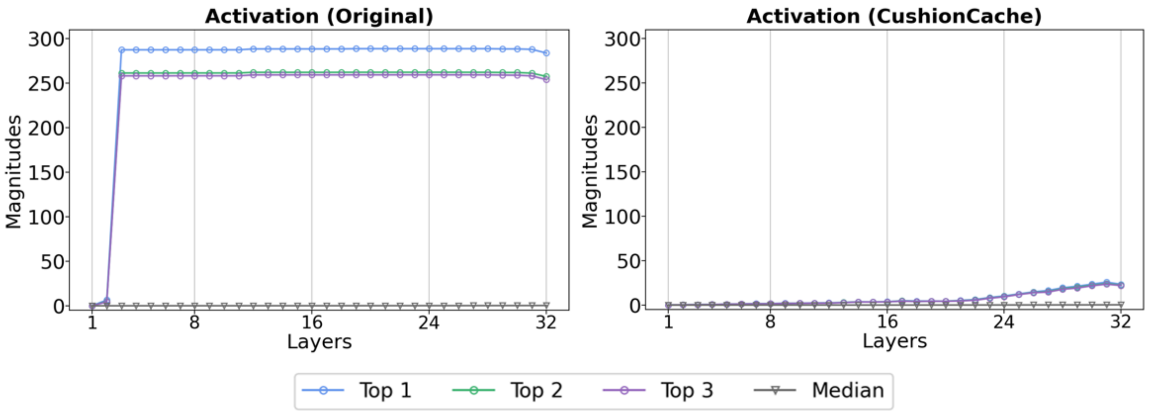

Figure 2: **Top-1/2/3 and median activation magnitudes at each layer of LLaMA3-8B.** The left panel shows the activations without CushionCache, having significant outliers except for initial layers. The right panel shows the activation with CushionCache, having significantly reduced outliers in every layers.

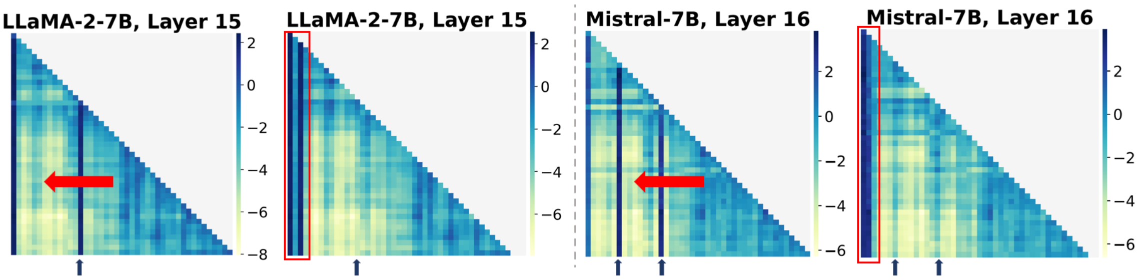

Figure 3: **Attention patterns before and after applying CushionCache in LLaMA3-8B and Mistral-7B.** The first and third panels show the attention patterns in models without CushionCache, where the attention sinks are quite prevalent in the generated token sequence. The second and fourth panels illustrate the attention patterns after inserting CushionCache. By adding the CushionCache, the attention is redirected toward the CushionCache tokens, preventing the attention sink from arising in the subsequent tokens.

In Fig. 2, we visualize the top-1/2/3 activations and median for each layer of LLaMA3-8B. The left panel plots the magnitude of the median and top-3 activations that occur during the standard operation of LLaMA3-8B. We observe that the median is almost zero, indicating that a significant fraction of all activations are close to zero, with only a few significantly large outliers. On the right panel, we plot the same values after applying the proposed CushionCache algorithm. We observe that the size of the top-3 activations have dramatically decreased, leading to a conclusion that CushionCache effectively removes the activation outliers.

6.2 Attention on CushionCache

In Fig. 3, we visualize the attention patterns of LLaMA2 and Mistral, before and after applying the CushionCache. Attention sinks, as identified by Xiao et al. (2024); Sun et al. (2024), are tokens that disproportionately attract attention. By inserting CushionCache, we observe that the CushionCache tends to dominate most of the attention from other tokens, removing the sinks in other tokens.

Model	Step 1	Step 2	Total Time
LLaMA2-7B	2.68 hours	3.34 hours	6.02 hours
LLaMA3-8B	12.09 hours	3.70 hours	15.79 hours
OPT-7B	1.38 hours	2.71 hours	4.09 hours

Table 6: **Wall-clock time for the search.** We use a server with four NVIDIA A6000 GPUs.

6.3 Time Needed to Search CushionCache

In Table 6, we report the wall-clock time spent for performing the greedy search and prefix tuning of CushionCache. We observe that the greedy prefix search can be quite time-consuming, highly dependent on the side of the embedding table; LLaMA3-8B has a large embedding table. Another observation is that the quantization-aware prefix tuning step takes relatively small time for all models.

7 Conclusion

In this paper, we present CushionCache, a novel approach for mitigating activation outliers in LLMs to improve activation quantization performance. Through extensive experiments, we demonstrate that CushionCache consistently enhances the performance of per-tensor activation quantization. Our analysis shows that CushionCache effectively reduces the magnitude of activation outliers and redirects attention sinks, leading to more uniform and quantization-friendly activations. In contrast with other approaches to faciliate activation quantization, CushionCache is the first—up to our knowledge—to fundamentally alter the activation distribution itself without extensive training, making activations easier to quantize.

Limitations

A limitation of our study is that our method is designed for LLMs with the decoder-only transformer structure. An extension to encoder-decoder LLMs (Raffel et al., 2020) may require further modifications to the algorithm. Another limitation is the lack of a principled mechanism to determine the hyperparameter τ, which decides when to stop adding new tokens. An extensive tuning may incur a non-negligible computational cost, especially when the target model is extremely large.

Ethics statement

All experimental results we provide in this paper is based on publicly available datasets and open-source models, whose intended use include research purposes. We have used an AI assistant for the grammar check.

Acknowledgements

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2023-00213710), and in part by the Korea government (MSIT) (No. RS-2024-00453301). JL also thanks Hong-Seok Kim and Radha Chandika for their generous support during his visit at Google.

References

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. 2024. QuaRot: Outlier-free 4-bit inference in rotated LLMs. *arXiv preprint 2404.00456*.

Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. 2021. Understanding and overcoming the challenges of efficient transformer quantization. In *Conference on Empirical Methods in Natural Language Processing*.

Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. 2023. Quantizable transformers: Removing outliers by helping attention heads do nothing. In *Advances in Neural Information Processing Systems*.

Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. 2024. Vision transformers need registers. In *International Conference on Learning Representations*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. LLM.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2023. A framework for few-shot language model evaluation.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. 2022. An empirical analysis of compute-optimal large language model training. In *Advances in Neural Information Processing Systems*.

Wei Huang, Xudong Ma, Haotong Qin, Xingyu Zheng, Chengtao Lv, Hong Chen, Jie Luo, Xiaojuan Qi, Xianglong Liu, and Michele Magno. 2024. How good are low-bit quantized LLaMA3 models? an empirical study. *arXiv preprint 2404.14047*.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7B. *arXiv preprint 2310.06825*.

Olga Kovaleva, Saurabh Kulshreshtha, Anna Rogers, and Anna Rumshisky. 2021. BERT busters: Outlier dimensions that disrupt transformers. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, et al. 2022. BLOOM: A 176B-parameter open-access multilingual language model. *arXiv preprint 2211.05100*.

Shiyao Li, Xuefei Ning, Luning Wang, Tengxuan Liu, Xiangsheng Shi, Shengen Yan, Guohao Dai, Huazhong Yang, and Yu Wang. 2024. Evaluating quantized large language models. *arXiv preprint 2402.18158*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In *Conference on Machine Learning and Systems*.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. Kivi: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the International Conference on Machine Learning*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint 1609.07843*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2024. OmniQuant: Omnidirectionally calibrated quantization for large language models. In *International Conference on Learning Representations*.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint 1909.08053*.

Mingjie Sun, Xinlei Chen, J Zico Kolter, and Zhuang Liu. 2024. Massive activations in large language models. *arXiv preprint 2402.17762*.

William Timkey and Marten van Schijndel. 2021. All bark and no bite: Rogue dimensions in transformer language models obscure representational quality. In *Conference on Empirical Methods in Natural Language Processing*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint 2307.09288*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*.

Xiuying Wei, Yunchen Zhang, Yuhang Li, Xiangguo Zhang, Ruihao Gong, Jinyang Guo, and Xianglong Liu. 2023. Outlier suppression+: Accurate quantization of large language models by equivalent and effective shifting and scaling. In *Conference on Empirical Methods in Natural Language Processing*.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the International Conference on Machine Learning*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. ZeroQuant: Efficient and affordable post-training quantization for large-scale transformers. In *Advances in Neural Information Processing Systems*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. OPT: Open pre-trained transformer language models. *arXiv preprint 2205.01068*.

A Additional experiments

In this section, we provide additional experiments that have been missing in the main script.

A.1 MMLU dataset

We have additionally evaluated the quantized model on MMLU dataset, which encompasses a much larger set of tasks including STEM (Table 7). The results suggest that CushionCache remains to be effective on MMLU as well.

Model	LLaMA2-7B	Mistral-7B	LLaMA3-8B
FP16	41.27	58.63	62.13
SmoothQuant-O3	23.76	23.62	25.32
+ CushionCache (ours)	38.06 (+14.30)	56.59 (+32.97)	58.99 (+33.67)
SmoothQuant-O2	27.88	25.66	30.94
+ CushionCache (ours)	40.45 (+12.59)	58.05 (+32.39)	60.59 (+29.65)
SmoothQuant-O1	40.76	58.70	61.88
+ CushionCache (ours)	41.65 (+0.89)	59.20 (+0.50)	61.55 (-0.33)

Table 7: **Results on MMLU.** We compare the results on the MMLU dataset.

A.2 Generation latency

We have measured the average latency of generating each token. We have experimented with W8A8-quantized LLaMA-3B, using the SmoothQuant kernel on a single A6000 GPU; not that this may not be the best hardware-optimized kernel for our hardware, but can be meaningful in terms of providing a comparison. We have used the prompts of length 500, and averaged over 1000 generated tokens. We compare both the time to the first token (TTFT; prefill phase) and the time per output token (TPOT; generation phase). in Table 8.

We observe that CushionCache only adds negligible latency, while enabling a much better adoption of the per-tensor static-range quantization techniques which provides a much faster decoding. In particular, we observe that adding CushionCache adds only 0.01–0.3ms in TTFT or TPOT, which is less than 0.5% of the total latency. Furthermore, as the CushionCache makes the faster option (e.g., per-tensor static) a viable option, it can even be viewed as enabling an overall speedup up to a few milliseconds.

LLaMA3-8B	TTFT (ms)	TPOT (ms)
Per-Tensor Static	78.01	48.71 ± 1.58
+ CushionCache (ours)	78.22	48.86 ± 0.55
Per-Tensor Dynamic	81.52	50.56 ± 0.71
+ CushionCache (ours)	81.56	50.93 ± 0.82
Per-Token Dynamic	83.35	51.75 ± 1.22
+ CushionCache (ours)	83.64	51.76 ± 0.95

Table 8: **Generation latency.** We measure the generation speed on LLaMA3-8B.

A.3 Other quantization methods

We have conducted additional experiments to combine with quantization algorithms other than SmoothQuant. In particular, we have conducted experiments on the following recent methods:

- AWQ-4bit (Lin et al., 2024): A recent weight-only quantization algorithm, for demonstrating that CushionCache effectively boosts the performance of weight quantization algorithms.
- QuaRot-4bit (Ashkboos et al., 2024): A recent weight+activation+cache quantization algorithm, for demonstrating that CushionCache works well with SOTA quantization algorithms.
- KIVI-2bit (Liu et al., 2024): A recent KV cache quantization algorithm, for demonstrating that the KV cache of the CushionCache-quantized model can be compressed well with KV cache quantization methods.

The results are given in Table 9, where we observe that the CushionCache is indeed versatile, being able to be combined well with many different quantization methods. Note that for KIVI, we measure the GSM8K results, as the original paper does not report perplexity.

LLaMA3-8B	WikiText-2 Perplexity
FP16	6.13
AWQ	6.18
+ CushionCache	6.15
+ Per-Cushion Static	8.40
+ Per-Cushion Static + CushionCache	7.01
QuaRot	8.21
+ CushionCache	7.41
LLaMA3-8B	**GSM8K (%)**
FP16	48.75
+ KIVI	42.61
Per-tensor Static	0.06
+ KIVI	0.03
+ KIVI + CushionCache	38.29

Table 9: **Other quantization methods.** We combine CushionCache with AWQ, QuaRot, and KIVI.