APEX-Q: Arbitrary-dimension Product-EXtension Quantization for Accelerated LLM Deployment (Student Abstract)

Yian Wang, Ye Qiao, Sitao Huang, Hyoukjun Kwon

University of California, Irvine, Department of EECS
Irvine, CA, 92697, USA
{yianw11, yeq6, sitao, hyoukjun.kwon}@uci.edu

Abstract

We present APEX-Q, a flexible product quantization framework for compressing large language models. Unlike prior multi-codebook quantization methods with fixed partitions, APEX-Q supports arbitrary-dimensional tensor quantization, better capturing weight redundancy. It achieves performance on par with 4-bit and 8-bit baselines, enables post-training quantization without retraining, and reveals key trade-offs across subvector dimensions, codebook sizes, and hardware efficiency. APEX-Q thus provides a unified, hardware-friendly approach to scalable LLM deployment.

Introduction

The rapid growth of large language models (LLMs) (Touvron 2023) has created an urgent need for efficient compression techniques that reduce storage, memory, and inference latency without compromising performance (Qiao et al. 2025b,a). Post-training quantization (PTQ) has emerged as the de facto standard due to its simplicity and hardware compatibility, enabling compression without retraining. Scalar quantization (SQ) is a common PTQ technique that converts high-precision formats like FP32 or FP16 to lower-precision formats such as INT8 or INT4. While effective, scalar quantization introduces trade-offs in compression and performance, demands careful calibration, and incurs runtime overhead due to explicit dequantization.

We present APEX-Q (**A**rbitrary-dimensional **P**roduct **Q**uantization), a novel PTQ framework designed as a drop-in replacement for scalar quantization. APEX-Q replaces traditional matrix multiplication with massive parallel table lookups, providing Multi-Codebook Quantization (MCQ) as an alternative to scalar quantization. APEX-Q achieves compression rates comparable to state-of-the-art INT4 schemes while maintaining competitive performance on language understanding, generation, and reasoning tasks. Unlike integer-based methods, APEX-Q operates entirely in floating-point, eliminating the need for runtime dequantization. Additionally, APEX-Q requires no calibration or fine-tuning, facilitating straightforward application to existing models.

APEX-Q introduces flexibility in compression granularity through varying design parameters, providing a continuum of compression-performance trade-offs. This enables finer control over model size and accuracy, making APEX-Q suitable for diverse deployment targets ranging from edge devices to large-scale inference servers.

APEX-Q Methods

In this section, we introduce APEX-Q, a framework that enables product quantization in LLMs.

APEX-Q Quantization

Product Quantization operates by partitioning the weight matrix $W \in \mathbb{R}^{d \times d'}$ into N_{ss} subspaces. The partitioning dimension, denoted as dim_{ss}, can be selected along either the row (d) or column (d') dimension. We use a binary variable dim to indicate this choice: if $dim = 0$, we split along the d (row) dimension; if $dim = 1$, we split along the d' (column) dimension. The remaining dimension is then treated as the datapoint dimension, dim_{dp}. The weight matrix is divided into N_{ss} subspaces along the selected subspace dimension. Thus, we have each weight subspace (W_{ss}) as:

$$W_{ss} \in \begin{cases} \mathbb{R}^{\frac{d}{N_{ss}} \times d'}, & \text{if dim} = 0 \\ \mathbb{R}^{d \times \frac{d'}{N_{ss}}}, & \text{if dim} = 1 \end{cases} \tag{1}$$

The size of each subspace is thus $SZ_{ss} = \frac{dim_{ss}}{N_{ss}}$.

After selecting the dimension of the subspace, we then determine the number of clusters K_s assigned to each subspace. This value is set by the number of bits b allocated for quantization, with the relationship $K_s = 2^b$. As a result, each subspace contains K_s clusters, and across all N_{ss} subspaces, the total number of clusters is $K_s \times N_{ss}$.

Within each subspace, we independently apply the K-Means clustering algorithm with K_s clusters along the datapoint dimension dim_{dp}. This approach ensures that the clustering process is specifically adapted to the structure of each subspace. For every subspace, K-Means produces a table of cluster centers $T^{ss}_{\text{cluster}} \in \mathbb{R}^{K_s \times SZ_{ss}}$ and an index table $T^{ss}_{\text{index}} \in \mathbb{R}^{dim_{dp}}$. These tables are then aggregated across all subspaces to obtain the final cluster center table $T_{\text{cluster}} \in \mathbb{R}^{N_{ss} \times K_s \times SZ_{ss}}$ and the index table $T_{\text{index}} \in \mathbb{R}^{N_{ss} \times dim_{dp}}$.

APEX-Q Inference

We developed a reconstruction-free inference method. Instead of recovering the full weight matrix, we compute a series of small lookup tables—one for each subspace—by performing vector-matrix multiplications between the input and

the relatively small cluster centers. This approach leverages the high memory and network-on-chip (NoC) bandwidth to reduce the number of computations. Since each subspace typically contains far fewer cluster centers than the number of original weight vectors, this method reduces the number of floating-point operations (FLOPs) by an average of 7×, as suggested by our experimental results.

Once the lookup tables are computed, the final outputs are generated by performing index-based lookups and accumulating results across subspaces. This reconstruction-free approach eliminates redundant computations inherent in traditional dense matrix multiplications, where many rows of the weight matrix may perform nearly identical operations due to sparsity and redundancy. By leveraging the clustered structure of quantized weights, APEX-Q reduces repeated multiply-accumulate operations with lightweight, parallel table lookups, significantly reducing FLOPs without increasing memory pressure.

We implemented this acceleration algorithm in CUDA C++. Experiment result shows that the end-to-end latency of APEX-Q matches its baseline, demonstrating possible future improvements. The detailed method is in Figure 1.

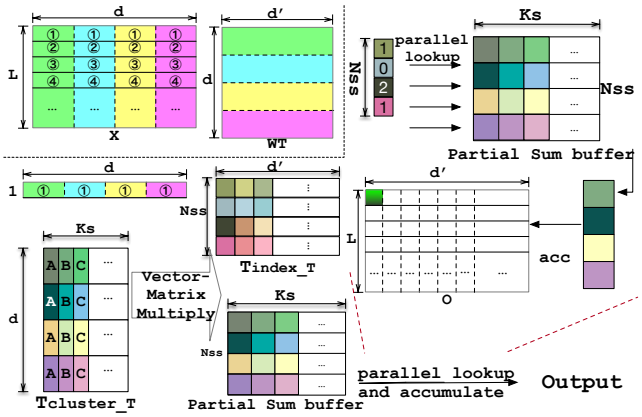

Figure 1: APEX-Q parallel inference method.

Preliminary Results

Baselines We target GPTQ (Frantar et al. 2022), AWQ (Lin et al. 2024), and SmoothQuant (Xiao et al. 2023) as current state-of-the-art in LLM quantization and serve as strong baselines for comparison with our proposed approach.

LLM Workloads We experiment on the LLaMA-2 (Touvron et al. 2023) models (7B, 13B), Qwen-2.5 (Yang et al. 2024) model (7B), and Mistral (Jiang et al. 2023) 7B model. We evaluate these models on the WikiText2 test set (Merity et al. 2016) and eight zero-shot commonsense reasoning tasks.

Results The compression rates together with benchmark results are summarized in Table 1. These results highlight that different variants of APEX-Q achieve distinct memory–performance trade-offs, reflecting their underlying design choices. In particular, some variants prioritize aggressive compression, significantly reducing memory usage at the cost of moderate accuracy degradation, while others strike a balance by preserving model quality with slightly lower compression. This diversity demonstrates the flexibility of APEX-Q in adapting to different deployment constraints, whether dominated by memory capacity, computational throughput, or accuracy requirements.

Method	CR	8 0-Shot Avg	PPL
Baseline	0%	64.07%	5.47
SmoothQuant	48.03%	63.65%	5.52
GPTQ-W3	77.16%	59.35%	6.77
GPTQ-W4	71.11%	63.51%	5.77
AWQ-W3	77.93%	60.91%	6.24
AWQ-W4	70.57%	63.83%	5.60
APEX-Q Flex	58.85%	63.25.%	5.65
APEX-Q Edge	75.94%	61.54%	6.04
APEX-Q Server	63.95%	63.76%	5.55

Table 1: Results of LLaMA2-7B model. CR: compression rate; PPL: perplexity.

Method	GEMV(ms)	GEMM(ms)
Baseline	1.719	3.254
APEX-Q	1.786	3.438

Table 2: Latency of the GEMM and GEMV computation.

The end-to-end latency comparison between APEX-Q and the FP16 baseline is presented in Table 2. The results demonstrate that the memory–performance trade-offs introduced by APEX-Q do not incur additional latency overhead. Instead, APEX-Q maintains execution times that are closely aligned with the FP16 baseline, confirming that its compression efficiency does not come at the expense of runtime performance. This latency proximity further highlights the practicality of APEX-Q, as it opens opportunities for the method to match—or even surpass—the inference speed of other quantization approaches, while simultaneously achieving significant reductions in memory footprint.

Conclusion

We presented APEX-Q, a quantization framework enabling higher-dimensional quantization with flexible design-space control. APEX-Q matches state-of-the-art performance and compression while providing greater flexibility across deployment constraints and hardware targets. Comprehensive design-space exploration on large language models demonstrates its effectiveness and robustness. To unlock efficiency, we implement CUDA kernels that use parallel table lookups in shared memory, bypassing weight reconstruction and directly computing matrix multiplications over cluster centers. The implementation reuses precomputed lookups, integrates with PyTorch, and delivers higher throughput than a naïve baseline, with latency comparable to optimized FP16 tensor-core matmul. Overall, APEX-Q is a promising, scalable, deployment-friendly approach for LLM quantization.

References

Frantar, E.; et al. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

Jiang, A. Q.; et al. 2023. Mistral 7b. arxiv. *arXiv preprint arXiv:2310.06825*, 10.

Lin, J.; et al. 2024. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. In *MLSys*.

Merity, S.; et al. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Qiao, Y.; Chen, Z.; Wang, Y.; Zhang, Y.; Deng, Y.; and Huang, S. 2025a. COBRA: Algorithm-Architecture Co-optimized Binary Transformer Accelerator for Edge Inference. *arXiv preprint arXiv:2504.16269*.

Qiao, Y.; Chen, Z.; Zhang, Y.; Wang, Y.; and Huang, S. 2025b. TeLLMe v2: An Efficient End-to-End Ternary LLM Prefill and Decode Accelerator with Table-Lookup Matmul on Edge FPGAs. *arXiv preprint arXiv:2510.15926*.

Touvron, H.; et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Touvron, H. o. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Xiao, G.; et al. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, 38087–38099. PMLR.

Yang, A.; et al. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.