QAQ: Query-adaptive Mixed-precision Quantization for Large Language Models

Shuxing Li*
University of Macau
Guangzhou Vocational University of Science and Technology
Guangzhou, China
li.shuxing@gkd.edu.cn

Huanrong Liu*
University of Macau
Macau, China
mc45100@um.edu.mo

Zelin Wang*
Avant Intelligence Ltd
Cambridge, United Kingdom
zelin.wang@avant-x.ai

Ruoyang Du
Avant Intelligence Ltd
Cambridge, United Kingdom
ruoyang.du@avant-x.ai

S H Lee
Avant Intelligence Ltd
Cambridge, United Kingdom
sharon.lee@avant-x.ai

Chunlin Tian
University of Macau
Macau, China
yc27402@um.edu.mo

Qingbiao Li†
University of Macau
Macau, China
qingbiaoli@um.edu.mo

Abstract

Large language models (LLMs) achieve strong performance, yet inference is still bounded by trade-offs between efficiency and accuracy. While quantization cuts memory and latency, it fails to flexibly accommodate heterogeneous inputs. We introduce Query-Adaptive Quantization (QAQ), a dynamic-precision scheme that decomposes model weights into bit-planes, employs a trainable router for query-conditioned precision selection, and supports on-demand CPU↔GPU loading. On Qwen3 and LLaMA-3.1, QAQ matches the accuracy of 8-bit baselines while reducing GPU memory footprint, with an associated latency overhead. These results suggest that QAQ offers a practical operating point on the efficiency–accuracy frontier for LLM inference.

1 Introduction

With the rapid adoption of large language models (LLMs) in natural language processing and other AI applications, the demand for efficient and reliable inference has become increasingly critical [Zhou et al., 2024]. Quantization has emerged as a practical means to improve inference efficiency by reducing memory footprint, bandwidth, and compute cost [Xiao et al., 2022, Frantar et al., 2022a, Dettmers et al., 2022]. However, aggressive low-bit quantization may incur non-negligible accuracy degradation [Frantar et al., 2022b, Lin et al., 2023, Xu et al., 2023]. Consequently, in LLM inference there is often an inherent efficiency–accuracy trade-off [Kurtic et al., 2024]. Striking a balance between efficiency and accuracy remains a central challenge. Existing approaches often rely on uniform quantization to compress models, but such a one-size-fits-all precision strategy fails to accommodate the diverse requirements of different inputs or sub-tasks, while maintaining multiple models at varying precisions further exacerbates memory overhead and system complexity.

*Equal contribution.
†Corresponding author.

ML For Systems workshop at Neural Information Processing Systems (NeurIPS 2025).

Addressing this challenge is particularly critical for enabling the deployment of LLMs on resource-constrained environments such as edge devices, where both efficiency and accuracy are essential for practical applications [Zeng et al., 2024].

Static low-bit quantization, while foundational for efficient LLM inference, employs a fixed quantization strategy that assigns uniform bit-widths (e.g., INT4) across all model layers regardless of input characteristics Dettmers et al. [2023], Frantar et al. [2022a]. This approach becomes suboptimal for real-world workloads where queries exhibit varying computational demands and activation magnitudes. Since static methods must accommodate the most challenging inputs to maintain accuracy, they adopt conservative quantization settings that over-provision precision for the majority of simpler queries, leading to unnecessary computational overhead. The fundamental issue stems from input-dependent activation sensitivity: outlier activations in certain inputs require higher precision to avoid significant quantization errors Xiao et al. [2022], Bondarenko et al. [2021], while typical inputs could be processed accurately with more aggressive quantization. This mismatch creates an inherent tension between optimizing for average-case efficiency and preserving worst-case accuracy, motivating the need for adaptive quantization approaches.

To resolve the core trade-off between efficiency and worst-case accuracy, we introduce Query-Adaptive Quantization (QAQ), a novel inference framework that dynamically adapts precision to each query. Our core innovation is a two-fold approach: a lightweight, query-sensitive policy network and a specialized memory management system. The policy network, trained to predict layer-wise sensitivity based on input features, serves as a dynamic router that assigns an optimal bit-width to each transformer layer. To enable this flexible routing, our system pre-stores multiple quantized versions of weights in CPU memory, dynamically materializing only the selected layers' precision profiles into the GPU cache. This co-designed algorithmic and system-level approach allows QAQ to tailor the model's precision to each unique query, delivering correctness with minimal computational overhead.

2 Query-Adaptive Quantization

Conventional inference paradigms are static: dense models activate all parameters, while quantized models use a fixed bit-width for all inputs Frantar et al. [2022a].During inference, all parameters and the entire network depth are usually activated regardless of the input, which further increases redundant computation. Some approaches attempt to mitigate this issue by introducing expert routing mechanisms [Cai et al., 2025], yet they still suffer from fixed precision and a rigid number of experts. This universal rigidity leads to a suboptimal trade-off between performance on hard inputs and efficiency on easy ones, failing to exploit the transient computational demands of the data.

We break this static paradigm with a co-designed, dynamic-precision inference system. Our contribution is the synergistic interplay of three components: (1) A trainable gating network acts as a per-query router, predicting the minimal sufficient bit-width for each network block based on input semantics. (2) To efficiently realize this dynamic precision, weights are pre-compressed into bit-planes, a format that decouples storage from bit-width and allows for flexible, on-the-fly precision reconstruction. (3) This is operationalized by a hardware-aware loading mechanism that streams only the router-selected bit-planes from CPU to GPU, minimizing both memory footprint and data transfer overhead.

The overall design is illustrated in Figure 1.

Bit-Plane Decomposition: We decompose each weight tensor into bit-planes, enabling flexible precision adjustment at block-level granularity. Let $W \in \mathbb{R}^{m\times n}$ be the original full-precision weight matrix. We represent it as:

$$W = \sum_{b=0}^{B-1} 2^b \cdot W^{(b)}, \quad W^{(b)} \in \{0,1\}^{m\times n} \tag{1}$$

where B is the maximum bit-width (e.g., 8), and $W^{(b)}$ denotes the b-th binary bit-plane. By selective loading only the most significant bit-planes during inference, the system can dynamically adjust precision without retraining.

Trainable Router: While bit-plane decomposition provides a flexible representation for multi-precision weights, we still need a mechanism to determine what precision inference should be

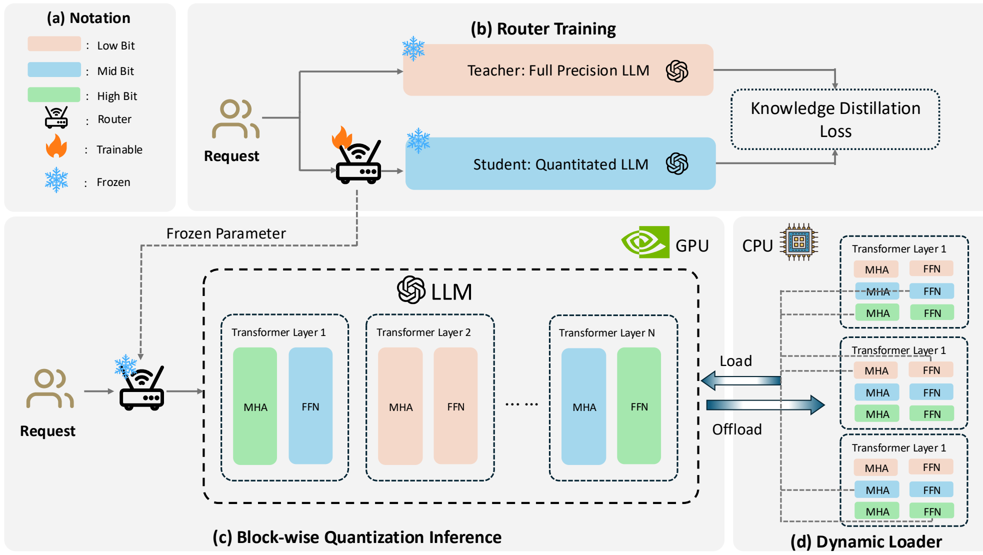

Figure 1: Framework of **QAQ**. (a) The notation distinguishes between low, mid, and high precision, indicating their association with router, trainable, and frozen parameters. (b) In Router Training, a full-precision teacher LLM instructs a quantized student LLM, with knowledge distillation loss guiding the training process. (c) During Block-wise Quantization Inference, transformer layers in the LLM are structured with Multi-Head Attention (MHA) and Feed-Forward Network (FFN) blocks. These layers are quantized at the block level for efficient resource management. (d) The dynamic loader facilitates the efficient transfer of transformer layers between CPU and GPU during inference, offloading and loading based on demand to optimize memory usage, ensuring high-performance inference with minimal computational overhead.

performed. To address this, we introduce a trainable router module. Here, precision selection is formulated as a soft routing decision, where the router assigns probabilities to candidate bit-widths based on query-dependent features. Given an input query representation x, the router computes an important score $s_j(x)$ for each block j, which reflects the sensitivity of that block to quantization. Formally, we define:

$$s_j(x) = f_\theta(h_j(x)) \tag{2}$$

where $h_j(x)$ is the hidden representation at block j, and $f_\theta(\cdot)$ is a trainable scoring function which is a lightweight MLP in our work.

The importance scores are normalized into a probability distribution over candidate bit-widths $b_1, \ldots, b_K$:

$$p_j(b \mid x) = \frac{\exp(\alpha \cdot s_j^b(x))}{\sum_{k=1}^{K} \exp(\alpha \cdot s_j^{b_k}(x))} \tag{3}$$

where α is a temperature parameter controlling the sharpness of the distribution. The expected precision at block j is then:

$$\hat{W}_j(x) = \sum_{b=1}^{K} p_j(b \mid x) \cdot W_j^{(b)} \tag{4}$$

where $W_l^{(b)}$ denotes the reconstructed weight matrix using the top-b bit-planes.

On-Demand Loading Mechanism: In conventional inference pipelines, all precision variants of weights are preloaded into GPU memory to guarantee fast access [Chen et al., 2018]. However, this approach results in a substantial memory footprint, since rarely used bit-planes remain resident in GPU memory [Qureshi et al., 2023]. On the other hand, placing the entire model in CPU memory or disk storage incurs prohibitive data transfer overhead during inference. To address this trade-off, we design an on-demand loading mechanism that dynamically transfers only the necessary bit-planes into GPU memory when they are required. Let $\mathcal{W} = W^{(b)}{}_{b=1}^{B}$ denote the set of bit-plane weights

Table 1: Accuracy, latency, and memory usgae comparison of **QAQ** against static quantization and full-precision baselines across LLaMA-3.1 and Qwen3 models. QAQ consistently maintains accuracy comparable to 8-bit quantization, while showing associated latency/memory trade-offs across on-demand modes.

Method	Hella-Swag	PIQA	ARC-E	ARC-C	Wino-Grande	WT2	PTB	Lat.(s)	Mem.(GB)
LLaMA3.1-8B									
Full FP16	78.90	81.18	81.10	53.50	73.48	6.24	9.01	75.12	23.12
Static 8-bit	59.99	79.98	81.65	51.45	73.56	6.24	9.01	56.61	13.26
Static 4-bit	59.29	80.30	81.44	50.09	72.93	6.71	9.10	55.72	13.26
QAQ (on-demand off)	**59.99**	**79.98**	**81.65**	**51.45**	**73.56**	**6.24**	**9.01**	**53.23**	**13.26**
QAQ (on-demand on)	**59.99**	**79.98**	**81.65**	**51.45**	**73.56**	**6.24**	**9.01**	**80.21**	**12.52**
Qwen3-4B									
Full FP16	68.42	74.97	78.49	53.92	66.06	13.64	18.74	44.23	14.23
Static 8-bit	68.45	74.81	78.32	53.92	65.98	14.83	18.75	38.91	9.25
Static 4-bit	66.78	75.14	76.94	52.99	63.22	14.83	20.25	38.04	9.25
QAQ (on-demand off)	**68.45**	**74.81**	**78.32**	**53.92**	**65.98**	**14.85**	**18.75**	**37.22**	**9.25**
QAQ (on-demand on)	**68.45**	**74.81**	**78.32**	**53.92**	**65.98**	**14.85**	**18.75**	**55.09**	**8.78**
Qwen3-8B									
Full FP16	74.95	77.80	80.89	56.48	67.72	9.72	13.54	72.32	19.32
Static 8-bit	74.92	77.74	80.85	56.57	67.80	10.30	13.54	65.64	11.42
Static 4-bit	73.98	77.80	80.18	57.42	66.06	10.30	14.41	64.91	11.42
QAQ (on-demand off)	**74.92**	**77.74**	**80.85**	**56.57**	**67.80**	**10.30**	**13.54**	**61.99**	**11.42**
QAQ (on-demand on)	**74.92**	**77.74**	**80.85**	**56.57**	**67.80**	**10.30**	**13.54**	**93.12**	**10.80**

Notes: Latency measured as end-to-end time for a single evaluation pass on WikiText-2. QAQ "on-demand off" disables CPU→GPU on-demand loading; "on-demand on" enables it.

for a given block. At time step t, the router determines a subset $\mathcal{S}_t \subseteq \mathcal{W}$ to be activated, based on query-dependent importance scores. GPU memory usage can be expressed as:

$$M_t = \sum_{W^{(b)} \in \mathcal{S}_t} \text{size}(W^{(b)}) \tag{5}$$

where $\text{size}(\cdot)$ denotes the storage cost of a bit-plane.

We define a loading operator $\mathcal{L}$ such that:

$$\tilde{W}_t = \mathcal{L}(\mathcal{S}_t), \quad \tilde{W}_t \subseteq \mathcal{W} \tag{6}$$

where $\tilde{W}_t$ is the set of weights available in GPU memory at time t. The operator $\mathcal{L}$ retrieves the required weights from CPU memory only when they are requested by the router.

3 Results

We evaluate QAQ on standard benchmarks using Qwen3-4B, Qwen3-8B, and Llama3.1-8B, comparing against fixed-4bit, fixed-8bit, and FP16 models. We test across different model scales to assess robustness, using Qwen3-4B, Qwen3-8B, and Llama3.1-8B. As shown in table 1, the results show that QAQ achieves accuracy nearly identical to the 8-bit baseline, with negligible degradation across all benchmarks. This suggests that our compression and routing strategies do not compromise model performance. We further report inference-related metrics, including latency on WikiText2 and GPU memory usage. When on-demand loading is disabled, QAQ achieves approximately 4.5% lower latency than fixed 4-bit baselines while keeping GPU memory consumption unchanged. With on-demand loading enabled, GPU memory usage decreases by 5.6% compared to static models, though latency increases by 41.7% due to synchronous weight transfers. This overhead reflects a current limitation: precision-adaptive bit-plane loading occurs sequentially, and computation stalls until higher-precision planes are fetched from CPU memory. Without asynchronous prefetching, these transfer costs cannot be hidden. Optimizing this pipeline to overlap communication and computation remains an important direction for future work.

References

Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. Understanding and overcoming the challenges of efficient transformer quantization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8998–9010, 2021.

Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. A survey on mixture of experts in large language models. *IEEE Transactions on Knowledge and Data Engineering*, 2025.

Xiaoming Chen, Danny Z Chen, and Xiaobo Sharon Hu. modnn: Memory optimal dnn training on gpus. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 13–18. IEEE, 2018.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022. doi: 10.48550/arXiv.2208.07339. NeurIPS 2022.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pretrained transformers. *arXiv preprint arXiv:2210.17323*, 2022a.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022b. doi: 10.48550/arXiv.2210.17323. ICLR 2023.

Eldar Kurtic, Alexandre Marques, Shubhra Pandit, Mark Kurtz, and Dan Alistarh. " give me bf16 or give me death"? accuracy-performance trade-offs in llm quantization. *arXiv preprint arXiv:2411.02355*, 2024.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023. doi: 10.48550/arXiv.2306.00978. MLSys 2024 Best Paper.

Zaid Qureshi, Vikram Sharma Mailthody, Isaac Gelado, Seungwon Min, Amna Masood, Jeongmin Park, Jinjun Xiong, Chris J Newburn, Dmitri Vainbrand, I-Hsin Chung, et al. Gpu-initiated on-demand high-throughput storage access in the bam system architecture. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 325–339, 2023.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. *arXiv preprint arXiv:2211.10438*, 2022. doi: 10.48550/arXiv.2211.10438. ICML 2023.

Zhaozhuo Xu, Zirui Liu, Beidi Chen, Yuxin Tang, Jue Wang, Kaixiong Zhou, Xia Hu, and Anshumali Shrivastava. Compress, then prompt: Improving accuracy-efficiency trade-off of llm inference with transferable prompt. *arXiv preprint arXiv:2305.11186*, 2023.

Binrui Zeng, Bin Ji, Xiaodong Liu, Jie Yu, Shasha Li, Jun Ma, Xiaopeng Li, Shangwen Wang, Xinran Hong, and Yongtao Tang. Lsaq: Layer-specific adaptive quantization for large language model deployment. *arXiv preprint arXiv:2412.18135*, 2024.

Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, et al. A survey on efficient inference for large language models. *arXiv preprint arXiv:2404.14294*, 2024.