AQUAPIC: BOOSTING ON-DEVICE LLM QUANTIZATION VIA PREFETCHING INPUT-WISE COMPENSATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Post-training quantization (PTQ) enables efficient local LLM inference, but aggressive quantization can substantially degrade accuracy. Recovering this lost accuracy typically requires additional correction data, but the gains are constrained by limited device memory budget. Prefetching correction data from second-tier memory can overcome this capacity limit, yet limited transfer bandwidth constrains their size, making compact low-rank compensation particularly suitable for this setting. We introduce input-partition-wise compensation (IPC), which exploits additional second-tier capacity by storing specialized low-rank compensators for different input regions without increasing the amount of data fetched per input. Building on IPC, AquaPIC combines output-error-aligned clustering, shared block-level partitions, and a lightweight residual-based partition estimator to select and prefetch compensators while the preceding block executes. Across two LLMs and two quantization methods, AquaPIC improves zero-shot accuracy under 2-bit quantization by 6.8 percentage points on average over the strongest same-rank baselines, while incurring only 2.27% average latency overhead across discrete GPU and unified memory setups.

1 INTRODUCTION

Local deployment of large language models (LLMs) has gained increasing attention due to its advantages in data privacy (Wang et al., 2025), low latency (Xu et al., 2024), and personalization (Li et al., 2025). These benefits enable applications ranging from Vision–Language–Action model-powered robotics (Kim et al., 2024a) to personalized LLM agents (Li et al., 2024), where accurate and responsive local inference is critical. Despite these advantages, achieving both low latency and high accuracy remains difficult under local GPU constraints. Local LLM inference is memory-bound in two ways: the small per-device batch size makes token generation bandwidth-bound, while the lack of multi-GPU scale-out caps memory capacity, limiting the deployable model size and thus accuracy.

Post-training quantization (PTQ) addresses these constraints by reducing parameter precision without retraining the model, lowering both memory footprint and memory traffic (Frantar et al., 2023; Lin et al., 2024; Tseng et al., 2024). This reduction enables larger models to fit within constrained local GPU memory while also decreasing weight-fetch overhead. Any device memory left over after quantization can be used to further improve accuracy. Techniques such as sparse outlier preservation (Dettmers et al., 2023; Lee et al., 2024a), mixed-precision quantization (Li et al., 2023; Huang et al., 2024; Kwon et al., 2025), and low-rank compensation (Yao et al., 2023; Zhang et al., 2024; 2025) use this remaining memory to retain or add higher-fidelity information. Each of them, however, requires device memory to store the correction data and adds latency to apply it during inference. For on-device LLMs under strict memory budgets, the former is the binding constraint: the achievable accuracy gain is bounded by how much correction data fits in device memory.

To overcome this limitation, a promising approach is to prefetch corrective data from secondary-tier storage (e.g., host memory or SSD) on demand. By keeping only the corrective data required for the currently executing attention/FFN block in device memory, this approach lets the total amount of corrective data scale with secondary-tier capacity rather than device memory, enabling far larger accuracy gains. Prefetching, however, turns the capacity limit into a transfer limit. The corrective data for the next block must arrive within the current block's execution time, over a link (e.g., PCIe)

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

with one to two orders of magnitude less bandwidth than device memory. This caps the payload that can be fetched per block.

Low-rank compensation (Yao et al., 2023; Zhang et al., 2024; 2025) exhibits properties well matched to prefetching under a tight payload budget. It expresses the correction to a quantized weight matrix as the product of two thin dense matrices, so the payload is small and directly controlled by the rank. It is also applied through dense, homogeneous GPU operations, adding negligible latency to the block's computation. However, a single compensator per layer offers limited scope for exploiting additional second-tier capacity, since increasing its rank also increases transfer volume and can exceed the prefetching window. This motivates scaling the number of stored compensators rather than their individual rank, while fetching only the selected compensator for each layer and input.

To this end, we propose Augmenting Quantization with Prefetched Input-wise Compensation (AquaPIC), a post-training framework for input-adaptive low-rank compensation whose compensators reside in secondary-tier storage and are prefetched on demand in local quantized LLM inference. AquaPIC exploits the input dependence of quantization-induced output errors by partitioning the input space through output-error-aligned clustering and fitting one compensator per partition. A lightweight estimator predicts the next block's partition from the current residual activation so that the selected compensator is prefetched while the current block executes and its transfer stays off the decoding critical path. The PTQ base weights remain frozen throughout and both the compensators and the partition estimator are fitted on a small calibration set without any fine-tuning of the LLM.

Our contributions are summarized as follows:

- **Scaling low-rank compensation for input-adaptive prefetching.** To exploit additional second-tier capacity, we explore *input-partition-wise compensation* (IPC), a post-training scheme that specializes low-rank corrections to different input partitions without increasing the per-input compensator payload at a fixed rank and precision.
- **Practical realization of IPC for local LLM inference.** We develop AquaPIC, combining output-error-aligned clustering, shared block-level partitions, and a lightweight partition estimator based on residual activations to select and prefetch the required compensators by overlapping with block computation.
- **Extensive validation of AquaPIC.** Across two LLMs quantized to 2 and 3 bits using QuIP# (Tseng et al., 2024) and GuidedQuant (Kim et al., 2025), AquaPIC outperforms prior low-rank compensation methods in most settings at the same rank while using smaller compensator payloads. AquaPIC delivers an average 6.8 percentage-point zero-shot accuracy gain over each setting's strongest low-rank compensation baseline at 2 bits, with only 2.27% average decoding latency overhead across discrete-GPU and unified-memory setups.

2 Background

2.1 Memory Bottlenecks in Local Inference

Local LLM inference is decode-dominated and typically runs at single-to-low batch sizes; consequently, inference latency largely depends on the amount of memory traffic required to fetch model parameters and the available memory bandwidth (Frantar et al., 2023; Lin et al., 2024). Key challenges in local LLM inference arise from the hardware limitations of local GPUs. First, compared to data-center accelerators (NVIDIA, 2023), local GPUs provide significantly lower memory bandwidth (NVIDIA, 2022), often by an order of magnitude. As a result, LLM execution on local GPUs experiences substantially increased inference latency, which emerges as a primary bottleneck for local LLM usability. Second, limited GPU memory capacity restricts the size of deployable models, directly constraining achievable model accuracy. When model parameters do not fully reside in GPU memory, inference requires host-to-device transfers, whose bandwidth is another order of magnitude lower than device memory bandwidth (PCI-SIG, 2017). As a result, streaming large amounts of weights can therefore substantially increase decoding latency, particularly when these transfers cannot be fully overlapped with computation.

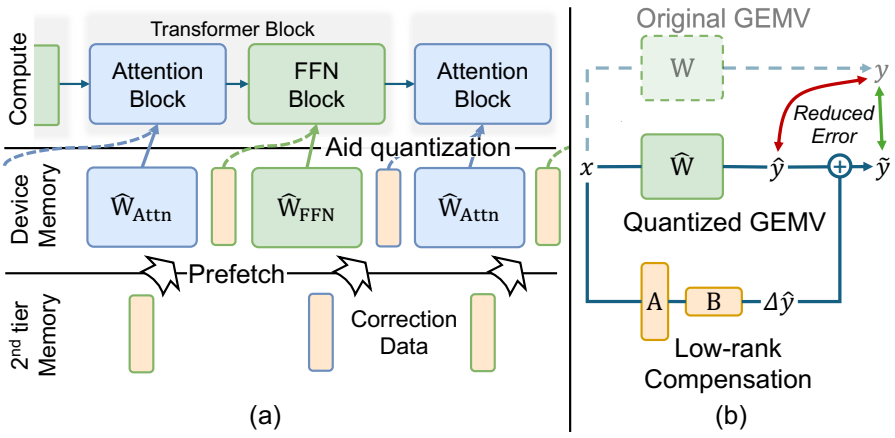

Figure 1: (a) Prefetching correction data in LLMs. (b) Low-rank compensation.

2.2 Quantization for Memory Efficiency

Quantization is a widely adopted technique for alleviating memory bandwidth and capacity bottlenecks. By reducing numerical precision, it compresses model parameters and lowers the overall memory footprint. Although quantization introduces accuracy degradation, the resulting gains in memory efficiency enable faster inference (Frantar et al., 2023; Lin et al., 2024; Park et al., 2024; Tseng et al., 2024) and allow the deployment of larger, more accurate models. While both weights and activations can be quantized, weight-only quantization is particularly well suited to the single-to-low-batch decoding scenario: it reduces the dominant source of memory traffic without introducing activation quantization error (Frantar et al., 2023; Lin et al., 2024; Kim et al., 2024b).

Practical on-device LLM deployments must also account for memory used by other applications, as a local LLM shares the device memory with foreground and background applications (Yu et al., 2026; Han et al., 2022). Consequently, the memory budget available to an LLM can be substantially smaller than the device's nominal capacity. These constraints further motivate aggressive weight quantization to keep model weights resident in device memory while reserving capacity for other workloads.

To compensate for the errors induced by aggressive quantization, methods such as sparse outlier preservation (Dettmers et al., 2023; Lee et al., 2024a), mixed-precision quantization (Li et al., 2023; Huang et al., 2024; Kwon et al., 2025), and low-rank compensation (Yao et al., 2023; Zhang et al., 2024; 2025) were proposed. These methods keep additional higher-fidelity information along with the quantized weights. However, as such data must reside in device memory, the accuracy gains of these methods are constrained by the available capacity. Thus, further accuracy recovery under a fixed device-memory budget requires exploiting memory capacity beyond the GPU.

2.3 Prefetching Opportunity in LLM

LLMs execute attention and feed-forward network (FFN) blocks sequentially, providing a natural window to prefetch data for an upcoming block while the current block executes. When model weights reside in GPU memory, decoding does not require host-to-device weight transfers, leaving PCIe bandwidth available to fetch auxiliary data from a second-tier memory, such as host memory or SSD storage. For quantized inference, this window can be used to prefetch correction data for the subsequent block during attention execution, helping recover accuracy lost to quantization, as illustrated in Figure 1(a). Keeping correction data in second-tier memory and buffering only the prefetched portion on the GPU limits the additional GPU memory footprint.

As the decoding latency is dominated by weight fetching memory traffic, the prefetching window can be approximated by the time required to read the current block's quantized weights from GPU memory. Let S_{Q} denote the size of the quantized weights of the block in bytes and BW_{GPU} the GPU memory bandwidth; the window duration is then $T \approx S_{\mathrm{Q}}/BW_{\mathrm{GPU}}$. The amount of auxiliary data that can be transferred during this window is approximately $S_{\mathrm{aux}} \approx T \cdot BW_{\mathrm{Link}}$, where BW_{Link}

denotes the PCIe transfer bandwidth. Since PCIe bandwidth is substantially lower than GPU memory bandwidth (e.g., $\sim 1/17\times$ for RTX 4060), the available transfer budget is only a small fraction of the quantized weight size. An effective compensation mechanism must therefore provide meaningful error reduction with compact auxiliary data and little additional GPU computation.

2.4 Low-Rank Compensation

Low-rank compensation provides a compact and computationally efficient correction by approximating the residual between the original ($\mathbf{W}$) and quantized weights ($\hat{\mathbf{W}}$) as the product of two low-rank factors ($\mathbf{A}, \mathbf{B}$). As illustrated in Figure 1(b), these factors form an additional compensation path ($\Delta\hat{y} = \mathbf{BA}\boldsymbol{x}$) alongside the quantized GEMV path ($\hat{y} = \hat{\mathbf{W}}\boldsymbol{x}$) in the linear layer. The compensated result can reduce the output error compared to the quantized GEMV result.

For a sufficiently small rank, these factors require little additional storage, making them suitable for compensation under tight data-transfer budgets. Low-rank compensation also uses dense matrix operations with regular memory access patterns, allowing efficient GPU execution.

Existing methods differ in how they formulate the objective function and solve it. ZeroQuant-V2 (Yao et al., 2023) applies truncated singular value decomposition (SVD) directly to the weight quantization residual. LQER (Zhang et al., 2024) uses a heuristic activation-based scaling of the residual before SVD to guide low-rank reconstruction, and QERA (Zhang et al., 2025) formulates compensation as minimizing the expected layer output error and derives both an exact and approximate solution that accounts for input statistics.

3 Scaling counts instead of ranks: Input-partition-wise Compensation

To fully exploit the abundant secondary storage under limited link bandwidth, simply increasing the compensator rank is impractical, as it also increases the per-input transfer volume. A more practical alternative is to scale the number of stored compensators while fetching only one for each input, thereby increasing secondary-storage usage without increasing the transfer volume.

The compensation path $\mathbf{BA}\boldsymbol{x}$ targets the output error $\Delta\mathbf{W}\boldsymbol{x}$, which depends not only on the weight residual but also on the input. Accordingly, prior work has shown that accounting for input statistics improves low-rank compensation (Zhang et al., 2024; 2025) over approximating the weight residual alone (Yao et al., 2023). As activation statistics vary across inputs, a compensator fitted to a narrower input region can correct the errors within that region more precisely at the same rank. This provides a natural way to scale the number of compensators: specialize different compensators to different input regions while keeping the per-input compensator payload fixed.

Motivational Intuition. Consider the weight of a linear layer $\mathbf{W} \in \mathbb{R}^{M\times K}$, calibration data $\mathbf{X} \in \mathbb{R}^{K\times N}$ of N input tokens, and a quantized weight $\hat{\mathbf{W}} \in \mathcal{Q}$, where $\mathcal{Q} \subset \mathbb{R}^{M\times K}$ is the feasible set determined by the quantization scheme and the bitwidth. Then, the layer-wise output error, which is commonly used as a proxy objective for quantization (Frantar et al., 2023; Tseng et al., 2024; Lin et al., 2024), is formulated as:

$$\mathcal{L} = \|\mathbf{WX} - \hat{\mathbf{W}}\mathbf{X}\|_F^2 = \|\Delta\mathbf{WX}\|_F^2 = \mathrm{tr}(\Delta\mathbf{WH}\Delta\mathbf{W}^\top) \tag{1}$$

where $\Delta\mathbf{W} = \mathbf{W} - \hat{\mathbf{W}}$ and $\mathbf{H} = \mathbf{XX}^\top$.

In the context of low-rank compensation, the goal is to find compensators $\mathbf{A} \in \mathbb{R}^{R\times K}$ and $\mathbf{B} \in \mathbb{R}^{M\times R}$ that will minimize the following:

$$\mathcal{L} = \|\mathbf{WX} - (\hat{\mathbf{W}} + \mathbf{BA})\mathbf{X}\|_F^2 = \|\mathbf{RX}\|_F^2 = \mathrm{tr}\left(\mathbf{RHR}^\top\right) \tag{2}$$

where $\mathbf{R} = \Delta\mathbf{W} - \mathbf{BA}$.

Suppose that the calibration data $\mathbf{X}$ is partitioned into $\mathbf{X}_1 \in \mathbb{R}^{K\times N_1}$ and $\mathbf{X}_2 \in \mathbb{R}^{K\times N_2}$, where $N = N_1 + N_2$. Under this partitioning, the Hessian matrix can be decomposed into the sum of the corresponding Hessians:

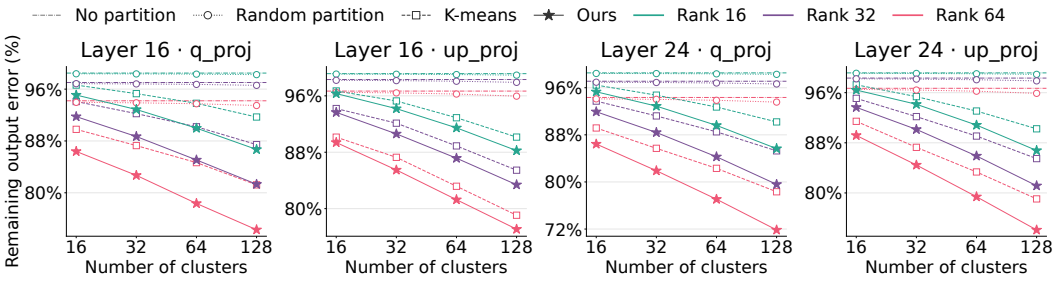

Figure 2: Output error trends for different input partitioning methods. Measured on Llama-3.1-8B-Instruct.

$$\mathbf{H} = \mathbf{H}_1 + \mathbf{H}_2 = \mathbf{X}_1\mathbf{X}_1^\top + \mathbf{X}_2\mathbf{X}_2^\top \tag{3}$$

If two different sets of compensators $(\mathbf{A}_1, \mathbf{B}_1)$ and $(\mathbf{A}_2, \mathbf{B}_2)$ are optimized for each split, with $\mathbf{R}_1 = \Delta\mathbf{W} - \mathbf{B}_1\mathbf{A}_1$ and $\mathbf{R}_2 = \Delta\mathbf{W} - \mathbf{B}_2\mathbf{A}_2$, the compensation objective decomposes into Equation 4.

$$\min_{\mathbf{A}_1,\mathbf{B}_1} \operatorname{tr}\left(\mathbf{R}_1\mathbf{H}_1\mathbf{R}_1^\top\right) + \min_{\mathbf{A}_2,\mathbf{B}_2} \operatorname{tr}\left(\mathbf{R}_2\mathbf{H}_2\mathbf{R}_2^\top\right) \tag{4}$$

The objective in Equation 4 is always less than or equal to the joint objective $\min_{\mathbf{A},\mathbf{B}} \mathcal{L}$, since the latter is equivalent to the former but with an additional constraint $(\mathbf{A}_1, \mathbf{B}_1) = (\mathbf{A}_2, \mathbf{B}_2)$.

This result naturally generalizes to cases with finer partitions of the input space, where a unique compensator set is assigned to each partition. As the partitioning becomes finer, the output error is guaranteed to be non-increasing, leading to a monotonic reduction in the overall output error.

Finding Compensator. Given the Hessian, obtaining each cluster's low-rank compensator can be done analytically. Since $\mathbf{H}_c$ is positive semi-definite but not necessarily positive definite, a small diagonal damping term $\lambda\mathbf{I}$ is added before computing its Cholesky factorization (Frantar et al., 2023; Fukaya et al., 2020), i.e., $\mathbf{H}_c + \lambda\mathbf{I} = \mathbf{L}_c\mathbf{L}_c^\top$. We seek rank-$R$ compensators $\mathbf{A}_c \in \mathbb{R}^{R\times K}$ and $\mathbf{B}_c \in \mathbb{R}^{M\times R}$ whose product approximates the quantization residual under this regularized Hessian. Writing the remaining residual as $\mathbf{R}_c = \Delta\mathbf{W} - \mathbf{B}_c\mathbf{A}_c$, the objective becomes:

$$\mathcal{L}_c = \operatorname{tr}\left(\mathbf{R}_c\mathbf{L}_c\mathbf{L}_c^\top\mathbf{R}_c^\top\right) = \|\mathbf{R}_c\mathbf{L}_c\|_F^2 = \|\Delta\mathbf{W}\mathbf{L}_c - \mathbf{B}_c\mathbf{A}_c\mathbf{L}_c\|_F^2 . \tag{5}$$

Because $\mathbf{L}_c$ is invertible, minimizing this objective is equivalent to finding the best rank-R approximation to $\Delta\mathbf{W}\mathbf{L}_c$ in the Frobenius norm, which can be found by applying singular value decomposition and retaining the R largest singular values and their corresponding singular vectors:

$$\begin{gathered}\Delta\mathbf{W}\mathbf{L}_c = \mathbf{U}_c\mathbf{\Sigma}_c\mathbf{V}_c^\top,\ \mathbf{U}_{c,R} = \mathbf{U}_c[:,:R],\ \mathbf{\Sigma}_{c,R} = \mathbf{\Sigma}_c[:R,:R],\ \mathbf{V}_{c,R}^\top = \mathbf{V}_c^\top[:R,:] \\ \mathbf{B}_c = \mathbf{U}_{c,R}\sqrt{\mathbf{\Sigma}_{c,R}},\ \mathbf{A}_c = \sqrt{\mathbf{\Sigma}_{c,R}}\mathbf{V}_{c,R}^\top\,\mathbf{L}_c^{-1}.\end{gathered} \tag{6}$$

The resulting closed-form solution for the low-rank compensator is equivalent to that derived in QERA (Zhang et al., 2025).

4 AquaPIC: Realizing IPC for Prefetching in LLM

Building on IPC, we design AquaPIC, an augmentation framework for quantized LLM inference that realizes input-partition-wise compensation for prefetching.

Figure 3 provides an overview of AquaPIC. While additional implementation details are provided in Appendix A, this section focuses on four key challenges in realizing the benefits of IPC. First, the input space must be partitioned so that each partition's compensator meaningfully reduces the output error. Second, the partition associated with each input must be identified at runtime. Third, enabling prefetching requires predicting the partition before the corresponding block begins. Fourth, the partitioning granularity must balance layer-specific specialization with prefetching efficiency.

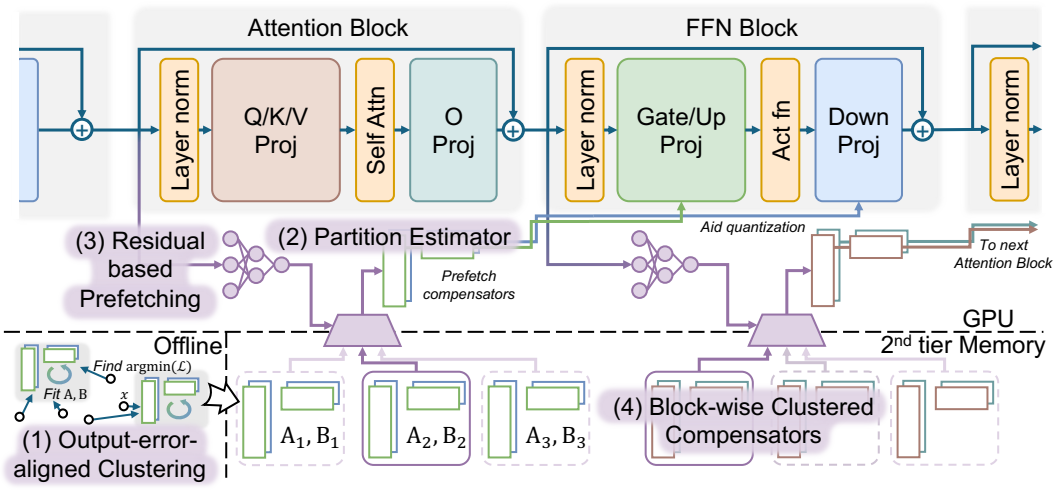

Figure 3: Overview of AquaPIC.

4.1 Output-Error-Aligned Clustering

Although the preceding analysis of Section 3 shows that partitioning cannot increase the optimal output error, the magnitude of the improvement depends on how the input space is partitioned. In particular, when the cluster Hessian $\mathbf{H}_c$ has a similar structure to the global Hessian $\mathbf{H}$, the resulting low-rank corrections tend to resemble the shared correction, yielding limited gains. Effective partitioning therefore requires clusters with distinct activation statistics that allow their compensators to specialize.

To align input partitioning with the output-error objective, we adopt a Lloyd-style procedure that alternates between cluster assignment and compensator optimization. Unlike standard K-means, which assigns inputs based on their Euclidean distance or cosine similarity to cluster centroids, our procedure assigns each input vector to the cluster whose compensator yields the lowest output error for that input. The update step then recomputes each cluster's Hessian from its assigned inputs and re-fits the corresponding low-rank compensator, replacing the centroid update in standard K-means.

Figure 2 compares how the output error decreases with increasing cluster counts under different partitioning methods. Random partitioning reduces the output error, which is consistent with the preceding analysis, but yields only marginal gains. K-means generally achieves lower error than random partitioning, indicating the importance of considering input characteristics when forming partitions. At the same compensator rank and number of partitions, our method can further lower the error than K-means, highlighting the benefit of aligning partitioning directly with the output-error objective. The detailed algorithm of the clustering process is in Appendix E.

4.2 Partition Estimator

Unlike standard K-means, our partitioning procedure assigns inputs according to the output error of each partition's compensator, rather than their distance to a centroid. Consequently, the resulting partitions are not characterized by centroids that can directly determine partition membership at runtime. We therefore fit a lightweight partition estimator to predict the partition associated with each input.

The estimator is a two-layer MLP, trained using the partition assignments obtained during clustering. For an input vector $\boldsymbol{x}$, it produces a probability distribution over partitions:

$$\mathbf{p}(\boldsymbol{x}) = \mathrm{softmax}(\mathbf{W}_2\,\mathrm{SiLU}(\mathbf{W}_1\boldsymbol{x}))\,, \tag{7}$$

where $\mathbf{W}_1$ and $\mathbf{W}_2$ are the learned weight matrices. At runtime, we select the partition with the highest predicted probability and apply its corresponding compensator. For Llama-3.1-8B, the hidden dimension for the estimator is 128, adding a storage overhead equivalent to 0.08 bits per model parameter.

Table 1 compares the top-1 partition assignment accuracy of the trained estimator with that of input centroid-based assignment for the output error-based partitions. The trained estimator significantly improves on the centroid-based assignment, allowing AquaPIC to realize the potential of IPC.

Table 1: Top-1 partition assignment accuracy (%) for 128 clusters. Measured using Llama-3.1-8B-Instruct.

Linear	Trained estimator	Input centroid
L0 `q_proj`	71.0	66.7
L0 `up_proj`	82.5	64.2
L16 `q_proj`	66.9	46.2
L16 `up_proj`	74.4	58.5

4.3 Prefetching Scheme

To make IPC practical under limited GPU memory, AquaPIC stores partition-specific compensators in high-capacity secondary memory and prefetches them into GPU memory before use. Therefore, prefetching must overlap with the current computation, but the hidden state needed for identifying the next block's input partition is not yet available while the current block is executing. Thus, AquaPIC must predict the partition early enough to initiate the corresponding transfer.

To do so, AquaPIC uses the residual available before the current block as a proxy for the next block's input. This approximation is motivated by the observation that residual connections in Transformers allow hidden states to change gradually across sublayers (Liu et al., 2023; Lee et al., 2024b). AquaPIC feeds this pre-block residual to the next block's partition estimator and prefetches the compensators associated with the predicted partition while the current block executes.

4.4 Clustering Granularity

Learning separate partitions and estimators for individual linear layers would allow finer-grained specialization, but each estimator would require the input activation of its corresponding layer. For later projections within an Attention/FFN block (e.g., O and Down projections), the corresponding intermediate activations become available only after earlier operations in the block have executed. We therefore form shared partitions at the Attention/FFN block level, so that a single prediction from this residual identifies the compensators needed throughout the next block. Each partition comprises one compensator per linear layer, allowing the entire set to be selected for prefetching while the current block executes.

During clustering, we fit these compensators together under shared block-level partition assignments, using the sum of the normalized mean squared errors (NMSEs) across the block's linear layers as the clustering objective. Let $\mathbb{L}$ denote the set of linear layers in a block and C the number of partitions. For linear layer ℓ, let $\mathbf{X}_\ell$ contain all calibration inputs and $\mathbf{X}_{\ell,c}$ contain those assigned to partition c under the shared block-level assignments. With $\Delta\mathbf{W}_\ell = \mathbf{W}_\ell - \hat{\mathbf{W}}_\ell$ and partition-specific compensator $\mathbf{B}_{\ell,c}\mathbf{A}_{\ell,c}$, we minimize

$$\mathcal{L}_{\text{cluster}} = \sum_{\ell\in\mathbb{L}} \text{NMSE}_\ell = \sum_{\ell\in\mathbb{L}} \sum_{c=1}^{C} \frac{\|(\Delta\mathbf{W}_\ell - \mathbf{B}_{\ell,c}\mathbf{A}_{\ell,c})\mathbf{X}_{\ell,c}\|_F^2}{\|\mathbf{W}_\ell\mathbf{X}_\ell\|_F^2}. \tag{8}$$

Each layer's error is normalized by its full-precision output energy over the calibration data, which balances the scale of each layer. The assignment step selects the partition that minimizes $\mathcal{L}_{\text{cluster}}$, and given these assignments, each linear layer's compensator is fitted independently using that layer's own Hessian matrix computed from the inputs assigned to the partition.

5 Evaluation

Models and Quantization. We use Llama-3.1-8B-Instruct (Meta, 2024) and Qwen3-4B-Instruct-2507 (Qwen Team, 2025) as the base LLMs. We apply 2- and 3-bit quantization to the models using the state-of-the-art weight-only quantization methods: QuIP# (Tseng et al., 2024) and GuidedQuant (Kim et al., 2025). For calibration, we sample 1,024 sequences of 4,096 tokens each from RedPajama (Together Computer, 2023). Sensitivity studies using different calibration sets are in Appendix D.1.

Baseline Methods. Under the same model and quantization configurations, we compare AquaPIC against prior low-rank compensation methods: ZeroQuant-V2 (Yao et al., 2023),

Table 2: Performance evaluation of AquaPIC: perplexity and downstream tasks. ZeroQuant-V2, LQER, and QERA are 16-bit in all ranks, while 32-rank Ours is 8-bit and 64-rank Ours is 4-bit.

Bits	Quant. Method	LRC Method	Llama-3.1-8B-Instruct						Qwen3-4B-Instruct-2507					
			Wiki(↓)		C4(↓)		Avg. 0-Shot (↑)		Wiki(↓)		C4(↓)		Avg. 0-Shot (↑)	
		Rank	32	64	32	64	32	64	32	64	32	64	32	64
16	–	–	7.22		11.48		74.0		10.06		17.20		75.7	
2	Guided Quant	–	40.01		44.60		44.1		51.31		42.53		32.5	
		ZQ-V2	29.59	25.91	36.83	34.78	49.5	52.7	40.98	34.30	37.07	33.97	38.0	41.1
		LQER	31.27	25.76	37.96	35.98	48.5	49.5	43.50	37.06	37.73	34.24	39.0	39.7
		QERA	29.33	24.73	36.09	33.66	50.5	52.8	40.08	32.51	35.82	32.57	39.7	43.8
		Ours	**26.75**	**22.54**	**29.07**	**26.33**	**52.0**	**53.2**	**37.17**	**32.22**	**28.64**	**26.45**	**46.7**	**49.3**
	QuIP#	–	13.45		22.09		42.4		14.24		22.59		59.4	
		ZQ-V2	13.31	13.20	21.83	21.59	43.1	44.0	13.93	13.73	22.16	21.92	59.9	59.7
		LQER	13.39	13.28	21.95	21.74	43.0	44.1	13.02	12.97	20.97	20.88	61.3	62.5
		QERA	13.30	13.13	21.86	21.64	42.2	43.5	13.87	13.54	22.27	21.89	61.1	62.0
		Ours	**10.60**	**10.29**	**17.56**	**17.04**	**59.4**	**60.4**	**12.48**	**12.21**	**20.29**	**19.96**	**64.9**	**66.2**
3	Guided Quant	–	8.52		14.09		71.2		12.25		19.10		61.0	
		ZQ-V2	8.42	8.37	13.94	13.83	71.5	71.8	**11.83**	11.59	18.96	18.86	63.7	67.5
		LQER	8.43	8.36	13.94	13.84	71.5	71.5	12.02	11.74	18.96	18.86	67.2	66.3
		QERA	8.40	8.33	13.92	13.82	71.3	71.5	**11.83**	**11.51**	18.95	18.86	67.4	66.3
		Ours	**8.34**	**8.25**	**13.76**	**13.64**	**72.0**	**72.0**	11.94	11.78	**18.79**	**18.77**	**68.7**	**69.6**
	QuIP#	–	8.24		13.67		70.9		10.92		18.40		71.8	
		ZQ-V2	8.24	8.23	13.63	13.60	70.5	70.9	10.95	10.91	18.36	18.28	71.6	71.9
		LQER	8.24	8.23	13.65	13.62	70.6	70.6	10.79	10.80	18.23	18.18	72.3	72.4
		QERA	8.23	8.22	13.64	13.62	70.7	70.8	10.96	10.98	18.41	18.42	71.6	71.8
		Ours	**7.96**	**7.93**	**13.09**	**13.03**	**71.5**	**71.4**	**10.67**	**10.62**	**18.02**	**17.95**	**72.6**	**72.7**

LQER (Zhang et al., 2024), and QERA (Zhang et al., 2025). The baseline low-rank compensators are not quantized and use the original 16-bit representation. Among the baselines, QERA also serves as the single-compensator counterpart to AquaPIC, as the closed-form solution for the output error-minimizing low-rank compensator is identical.

AquaPIC Configuration. For this evaluation section, AquaPIC is set to optimize 128 clusters per block. To accommodate AquaPIC's prefetching scheme, unlike the baselines, 8-bit symmetric quantization is applied for the 32-rank version of AquaPIC, while 4-bit OPTQ quantization is applied for the 64-rank version. The secondary memory usage for this setup is available in Appendix B.4, while the offline preparation cost is reported in Appendix B.5. Experiments using different cluster counts are in Appendix D.4.

5.1 Accuracy Evaluation of AquaPIC

Methodology. We evaluate the performance gains of AquaPIC in perplexity and downstream accuracy, under fixed transferred rank sizes. We first measure the perplexity on two datasets: WikiText2 (Merity et al., 2016) and C4 (Raffel et al., 2020). We then measure zero-shot accuracy on ARC (Clark et al., 2018), OpenBookQA (Mihaylov et al., 2018), MMLU (Hendrycks et al., 2021), LAMBADA (Paperno et al., 2016), and report the average accuracy. Detailed information on the evaluation setup is provided in Appendix B.

Results. Table 2 compares perplexity and average zero-shot accuracy at compensation ranks of 32 and 64. At the same rank, AquaPIC achieves the best results in the majority of the evaluated configurations. The gains are particularly pronounced under 2-bit quantization, with an average

Table 3: Time per output token (TPOT) on Llama-3.1-8B-Instruct with 2-bit GuidedQuant and QuIP#. Rank 32 and Rank 64 use 8-bit and 4-bit compensators, respectively.

Config		RTX 4060 + Host Memory			Jetson Orin NX + SSD		
		GPU Resident	Host Prefetch	Overhead	GPU Resident	SSD Prefetch	Overhead
Guided Quant	Rank	Uncompensated TPOT: 36.70 ms			Uncompensated TPOT: 150.94 ms		
	32	37.66 ms	38.61 ms	2.52%	153.63 ms	156.87 ms	2.11%
	64	38.01 ms	38.93 ms	2.42%	154.84 ms	157.89 ms	1.97%
QuIP#	Rank	Uncompensated TPOT: 40.11 ms			Uncompensated TPOT: 162.23 ms		
	32	41.13 ms	42.07 ms	2.29%	166.21 ms	168.55 ms	1.41%
	64	41.50 ms	42.42 ms	2.22%	167.46 ms	172.85 ms	3.22%

zero-shot accuracy gain of 6.8 percentage points. At 3 bits, the improvements are reduced due to the smaller headroom left by the base quantization methods. Yet, AquaPIC retains the best C4 perplexity and average zero-shot accuracy in these configurations.

5.2 Latency Evaluation

Methodology. We implement AquaPIC on top of gpt-fast (meta-pytorch, 2023), a highly optimized local LLM inference codebase utilizing CUDA graphs. We evaluate the decoding throughput overhead of AquaPIC on two setups: a discrete GPU setup with RTX 4060 (8GB) and a unified memory system setup with Jetson Orin NX Super (8GB). The former setup fetches the low-rank compensators from the host memory, while the latter setup fetches directly from SSD storage. We set the decoding length to 4096 tokens and measure the time per output token (TPOT) for each setup. We compare the measured TPOT against a baseline configuration where a single set of low-rank compensators is resident in the device memory, with no cluster selection methods enabled.

Results. Table 3 compares the decoding TPOT of AquaPIC against device-resident low-rank compensation. AquaPIC adds only 2.27% prefetching overhead relative to the device-resident low-rank compensation, and results in 5.11% total overhead over the uncompensated quantized baseline. This indicates that AquaPIC successfully overlapped the compensation prefetching with the computation, with only the minimal estimator overheads being visible in both the discrete GPU and unified memory setups.

5.3 Ablation Study

Table 4: Ablation study using Qwen3-4B-Instruct-2507 with 2-bit QuIP#.

Variant	Wiki		C4	
Rank	32	64	32	64
Oracle	12.46	12.19	20.16	19.83
+ Estimator	12.51	12.22	20.23	19.90
+ Prefetch	12.48	12.21	20.29	19.98
+ Quant.	12.48	12.21	20.29	19.96

To assess the impact of the approximation techniques in AquaPIC, we conduct an ablation study using Qwen3-4B-Instruct-2507 with 2-bit QuIP#. Table 4 reports the perplexity changes as we successively introduce the estimator, prefetching, and compensator quantization. The oracle routes each input to the compensator that minimizes the output error. The estimator and prefetching yield modest deviations from oracle perplexity as expected, while compensator quantization has a negligible effect. Overall, the approximation techniques retain most of the gains of IPC, while enabling efficient prefetching with low overhead.

6 Conclusion

We present AquaPIC, a framework that boosts aggressively quantized LLMs for local inference using input-partition-wise low-rank compensation. Output-error-aligned clustering, lightweight partition estimation, and residual-based prefetching enable specialized compensators to be fetched from second-tier memory within a fixed transfer budget. Across the evaluated models and quantization methods, AquaPIC improves perplexity and zero-shot accuracy over prior low-rank compensation methods in most configurations, with minimal latency overheads.

AI USE STATEMENT

In this work, we used generative AI tools to implement methods. We have not used generative AI tools to generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, assist with translation, clean and reformat dataset, support qualitative and thematic data analysis, or interpret results. Additionally, we used generative AI tools to create or edit software code, draft parts of a research paper, edit a research paper to improve readability, and propose a title or keywords for a research paper. We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

arXiv. Terms of use for arxiv apis - arxiv info. https://info.arxiv.org/help/api/tou.html, 2026.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Common Crawl Foundation. Common crawl - terms of use. https://commoncrawl.org/terms-of-use, 2024.

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint arXiv:2306.03078*, 2023.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. OPTQ: Accurate Quantization for Generative Pre-trained Transformers. In *International Conference on Learning Representations*, 2023.

Takeshi Fukaya, Ramaseshan Kannan, Yuji Nakatsukasa, Yusaku Yamamoto, and Yuka Yanagisawa. Shifted cholesky qr for computing the qr factorization of ill-conditioned matrices. *SIAM Journal on Scientific Computing*, 42(1):A477–A503, 2020.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

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. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2): 217–288, 2011.

Mingcong Han, Hanze Zhang, Rong Chen, and Haibo Chen. Microsecond-scale preemption for concurrent GPU-accelerated DNN inferences. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 539–558, Carlsbad, CA, July 2022. USENIX Association. ISBN 978-1-939133-28-1. URL https://www.usenix.org/conference/osdi22/presentation/han.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. 2021.

Appu Shaji Hicham Badri. Gemlite: Towards building custom low-bit fused cuda kernels, August 2024. URL https://mobiusml.github.io/gemlite_blog/. Accessed: 2026-09-23.

Wei Huang, Haotong Qin, Yangdong Liu, Yawei Li, Qinshuo Liu, Xianglong Liu, Luca Benini, Michele Magno, Shiming Zhang, and Xiaojuan Qi. Slim-llm: Salience-driven mixed-precision quantization for large language models. *arXiv preprint arXiv:2405.14917*, 2024.

Jinuk Kim, Marwa El Halabi, Wonpyo Park, Clemens JS Schaefer, Deokjae Lee, Yeonhong Park, Jae W. Lee, and Hyun Oh Song. Guidedquant: Large language model quantization via exploiting end loss guidance. In *International Conference on Machine Learning (ICML)*, 2025.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024a.

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. Squeezellm: dense-and-sparse quantization. In *Proceedings of the 41st International Conference on Machine Learning*, 2024b.

Sangwoo Kwon, Seong Hoon Seo, Jae W. Lee, and Yeonhong Park. Dp-llm: Runtime model adaptation with dynamic layer-wise precision assignment. In *Proceedings of the 39th Conference on Neural Information Processing Systems*, 2025.

Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. Owq: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 13355–13364, 2024a.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management. In *USENIX Symposium on Operating Systems Design and Implementation, (OSDI)*, 2024b.

Liang Li, Xingke Yang, Wen Wu, Hao Wang, Tomoaki Ohtsuki, Xin Fu, Miao Pan, and Xuemin Shen. Mobillm: Enabling llm fine-tuning on the mobile device via server assisted side tuning. *arXiv preprint arXiv:2502.20421*, 2025.

Shiyao Li, Xuefei Ning, Ke Hong, Tengxuan Liu, Luning Wang, Xiuhong Li, Kai Zhong, Guohao Dai, Huazhong Yang, and Yu Wang. LLM-MQ: Mixed-precision Quantization for Efficient LLM Deployment. In *The Efficient Natural Language and Speech Processing Workshop with NeurIPS*, 2023.

Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, et al. Personal llm agents: Insights and survey about the capability, efficiency and security. *arXiv preprint arXiv:2401.05459*, 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 on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6: 87–100, 2024.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Ré, and Beidi Chen. Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time. In *International Conference on Machine Learning (ICML)*, 2023.

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

Meta. Introducing llama 3.1: Our most capable models to date, 2024. URL `https://ai.meta.com/blog/meta-llama-3-1/`. Accessed: 2026-09-16.

meta-pytorch. gpt-fast: Simple and efficient pytorch-native transformer text generation. `https://github.com/meta-pytorch/gpt-fast`, 2023. GitHub repository, BSD-3-Clause License.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018.

NVIDIA. GeForce RTX 40 Series Graphics Cards, 2022. URL https://www.nvidia.com/en-us/geforce/graphics-cards/40-series/. Accessed: 2026-01-20.

NVIDIA. NVIDIA H100 GPU Whitepaper, 2023. URL https://resources.nvidia.com/en-us-hopper-architecture/nvidia-h100-tensor-c. Accessed: 2026-01-20.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 1525–1534, 2016.

Yeonhong Park, Jake Hyun, SangLyul Cho, Bonggeun Sim, and Jae W Lee. Any-precision llm: low-cost deployment of multiple, different-sized llms. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 39682–39701, 2024.

PCI-SIG. PCI Express Base Specification Revision 4.0, Version 1.0, 2017. URL https://pcisig.com/PCIExpress/Specs/Base/_4.0_1.0. Accessed: 2026-01-20.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Together Computer. Redpajama: An open source recipe to reproduce llama training dataset, 2023. URL https://github.com/togethercomputer/RedPajama-Data.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. QuIP$\#$: Even better LLM quantization with hadamard incoherence and lattice codebooks. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=9BrydUVcoe.

Xubin Wang, Zhiqing Tang, Jianxiong Guo, Tianhui Meng, Chenhao Wang, Tian Wang, and Weijia Jia. Empowering edge intelligence: A comprehensive survey on on-device ai models. *ACM Computing Surveys*, 57(9):1–39, 2025.

Jiajun Xu, Zhiyuan Li, Wei Chen, Qun Wang, Xin Gao, Qi Cai, and Ziyuan Ling. On-device language models: A comprehensive review. *arXiv preprint arXiv:2409.00088*, 2024.

Zhewei Yao, Xiaoxia Wu, Cheng Li, Stephen Youn, and Yuxiong He. Zeroquant-v2: Exploring post-training quantization in llms from comprehensive study to low rank compensation, 2023. URL https://arxiv.org/abs/2303.08302.

Hongseung Yu, Minsung Kim, Jongseok Park, and Kyunghan Lee. mzcache: On-device llm memory management under multitasking. *arXiv preprint arXiv:2609.01338*, 2026.

Cheng Zhang, Jianyi Cheng, George A Constantinides, and Yiren Zhao. Lqer: Low-rank quantization error reconstruction for llms. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Cheng Zhang, Jeffrey TH Wong, Can Xiao, George Constantinides, and Yiren Zhao. Qera: an analytical framework for quantization error reconstruction. In *International Conference on Learning Representations*, volume 2025, pp. 12531–12560, 2025.

A Further Details of AquaPIC

Pre-clustering Procedure. Before running the Lloyd-style procedure that minimizes the output error, we initialize the input partitions using K-means clustering. We first project the calibration input vectors into a 64-dimensional space using a random projection matrix whose entries are sampled from a Gaussian distribution, and then perform K-means in this projected space. The resulting assignments initialize the subsequent output-error-based clustering procedure. The projection is used only for this initialization; compensator optimization uses the original input vectors.

Fallback Compensator. For a configuration with C low-rank compensators, we optimize $C-1$ partition-specific compensators and reserve one as a shared fallback compensator. The fallback compensator is constructed using all calibration inputs and remains fixed throughout clustering, while the partition-specific compensators are updated using their assigned inputs. All C compensators are candidates during the output-error-based assignment step. This design is motivated by our observation that inputs near partition boundaries not only benefit less from specialized compensators themselves, but can also perturb the cluster Hessian and reduce the gains for other inputs in the same cluster. The shared fallback compensator provides an alternative for such inputs.

Estimator Training Details. After clustering, we evaluate each calibration input using the compensators of every partition. Let $e_{i,c}$ denote the aggregate output error for input i under partition c, computed as the sum of the linear-layer NMSEs described above. We convert these errors into a soft target distribution for training the partition estimator:

$$q_{i,c} = \frac{\exp(-e_{i,c}/T)}{\sum_{j=1}^{C} \exp(-e_{i,j}/T)}, \qquad c \in \{1, \ldots, C\}, \tag{9}$$

where $T > 0$ is the softmax temperature. This distribution assigns higher probability to partitions with lower output error, with smaller temperatures concentrating the probability mass on the lowest-error partitions. We train the estimator to predict this distribution, allowing the supervision to reflect the relative compensation quality of all partitions.

Reducing SVD cost. As the target rank of each compensator is substantially smaller than the dimensions of the corresponding weight matrix, we use randomized SVD (Halko et al., 2011) to reduce the computational cost of the repeated decompositions required during compensator optimization. Empirically, we observe negligible differences in compensation quality compared with using standard SVD.

Early Exiting. As clustering progresses, the reduction in output error per iteration tends to diminish. To avoid unnecessary iterations, AquaPIC monitors the fraction of calibration tokens whose partition assignments change relative to the previous iteration and terminates clustering when this fraction falls below a predefined threshold.

B Evaluation Details

B.1 Dataset Setup Details

For the calibration set, the RedPajama dataset is sampled and tokenized until the number of samples equals 1024. A text from the corpus is randomly sampled, and 4096 tokens from the text sample are concatenated to the calibration set. The text sample is discarded if it has fewer than 4096 tokens.

For perplexity measurement, we follow the methodology of Any-Precision LLM (Park et al., 2024). Specifically, the dataset text is concatenated and chunked into sizes of 2048. For downstream zero-shot tasks, the `lm-evaluation-harness` (Gao et al., 2024) framework is used with the chat format, as instruction-finetuned models are used for evaluation. For evaluating AquaPIC, the input sequences are interpreted as teacher-forced decoding phases. For the ARC dataset, the `arc_challenge_chat` variant with a slightly modified chat format is used. The format is as follows: "Given the following question and four candidate answers (A, B, C and D), choose the best answer. \nQuestion: {Question}\nA. {Choice A}\nB. {Choice B}\nC. {Choice C}\nD. {Choice D}\n\nAnswer: The best answer is." For OpenbookQA and MMLU, the same format as ARC is used with minimal changes. For LAMBADA, the default settings are used.

B.2 Documentation of Evaluation Data

Detailed Information of Data. Detailed information about the models and datasets used in the evaluation is as follows:

- Models
  - Qwen3-4B-Instruct-2507 (Qwen Team, 2025): Multilingual causal LM trained on a 36T-token corpus spanning 119 languages and domains, including code, STEM, reasoning, books, multilingual, and synthetic data.

  - Llama-3.1-8B-Instruct (Meta, 2024): Multilingual causal LM trained on a 15T-token corpus which mixes public online data.
- Datasets
  - WikiText-2 (Merity et al., 2016): English LM benchmark drawn from verified Wikipedia articles.
  - C4 (Raffel et al., 2020): Cleaned version of web crawl dataset of Common Crawl. The English variant is used.
  - RedPajama (Together Computer, 2023): Primarily English text generation dataset sourced from Common Crawl, C4, GitHub, arXiv, Wikipedia and StackExchange.
  - ARC (Clark et al., 2018): English grade-school multiple-choice science QA benchmark.
  - OpenBookQA (Mihaylov et al., 2018): English multiple-choice science QA benchmark for multi-step reasoning with open-book facts and commonsense knowledge.
  - MMLU (Hendrycks et al., 2021): English multiple-choice benchmark spanning knowledge and reasoning subjects across the social sciences, humanities, law, STEM, and other domains.
  - LAMBADA (Paperno et al., 2016): English word prediction benchmark extracted from BookCorpus.

License and Terms of Use. The license and terms of use for the models and datasets used in the evaluations are as follows:

- Models
  - Qwen3-4B-Instruct-2507 (Qwen Team, 2025): Apache-2.0
  - Llama-3.1-8B-Instruct (Meta, 2024): Llama 3.1 Community License
- Datasets
  - WikiText-2 (Merity et al., 2016): CC BY-SA 4.0
  - C4 (Raffel et al., 2020): ODC-BY
  - RedPajama (Together Computer, 2023): Composition of Common Crawl Foundation Terms of Use (Common Crawl Foundation, 2024), C4 License (ODC-BY), GitHub License (MIT, BSD, Apache), ArXiV Terms of Use (arXiv, 2026), Wikipedia Dataset License(GFDL, CC BY-SA 3.0), and StackExchange License (CC-BY-SA 4.0)
  - ARC (Clark et al., 2018): CC BY-SA 4.0
  - OpenBookQA (Mihaylov et al., 2018): Unknown
  - MMLU (Hendrycks et al., 2021): MIT
  - LAMBADA (Paperno et al., 2016): CC BY 4.0

B.3 Configuration for AquaPIC

The detailed configuration for AquaPIC used in the evaluations can be found in Table 5.

Table 5: Configuration parameters for AquaPIC

Configuration Parameter	Value
Lloyd-style Iteration Count	5
Ratio of Tokens with Changed Clusters for Early Stopping	1%
Estimator Hidden Dimension	128
Softmax Temperature for Estimator Training	0.01
Estimator Dropout Ratio	0.3
Estimator Training Learning Rate	3×10^{-3}
Estimator Training Weight Decay Rate	10^{-4}
Estimator Training Epoch	5
Damping Ratio for Hessian	0.01
Rank Overestimation for Randomized SVD	32
Iterations for Randomized SVD	4

B.4 Secondary Memory Usage

AquaPIC assumes access to ample second-tier memory and therefore uses more storage than prior low-rank compensation methods to accommodate cluster-specific compensation matrices. Table 6 reports the secondary memory footprint with 128 clusters across different compensation ranks and compensator precisions. These figures account for secondary memory usage only and are separate from device memory consumption.

Table 6: Secondary memory usage of AquaPIC with 128 clusters at different compensation ranks and compensator precisions.

Model	Rank	16-bit	8-bit	4-bit
Llama-3.1-8B-Instruct	64	41GiB	21GiB	11GiB
	32	21GiB	11GiB	6GiB
Qwen3-4B-Instruct-2507	64	32GiB	17GiB	9GiB
	32	16GiB	9GiB	5GiB

B.5 Offline Cost

The offline process of AquaPIC is parallelized across multiple GPUs. Using a workstation equipped with eight RTX 3090 GPUs, the process requires roughly 9 hours for Llama-3.1-8B-Instruct and 7.5 hours for Qwen3-4B-Instruct-2507.

C Runtime Implementation Details

For the quantized low-rank compensation path of AquaPIC, the GemLite (Hicham Badri, 2024) kernel is used for efficient W8A16 or W4A16 GEMV. The quantized GEMV paths for the linear layers utilize the state-of-the-art kernels for each quantization method: GuidedQuant uses the highly optimized non-uniform scalar kernel from Any-Precision LLM (Park et al., 2024), and QuIP# uses the official E8P12 GEMV kernel.

AquaPIC captures an LLM decoding iteration in a single CUDA graph. Two reusable compensator buffers, one for attention and one for FFN, serve all layers. Prefetching overlaps with the preceding block, with graph dependencies enforcing proper buffer usage.

On discrete GPU setups, the compensators are preloaded into mapped page-locked host memory. A custom CUDA kernel fuses estimator argmax with gathering the selected record into device memory, avoiding per-block CPU coordination. Gather kernels execute on two auxiliary CUDA streams. Cross-stream dependencies are expressed through CUDA events, forming graph dependencies that enforce safe buffer reuse and completion of each gather before compensation.

On unified memory setups, two persistent CPU workers perform SSD reads. After the GPU identifies the cluster, it publishes the cluster ID and a request flag in mapped page-locked host memory. CPU workers poll the request flag and publish a ready flag after completing the read. The ready flag is read by the GPU by a batch memory operation node, `cuGraphAddBatchMemOpNode`. Explicit graph dependencies order publication before the wait and compensation after it, preserving graph replay without GPU polling kernels.

SSD reads use `O_DIRECT` with 4-KiB-aligned records and buffers. Buffers are registered with `io_uring` and reused for batched fixed-buffer reads, using 64-KiB chunks and a queue depth of 64 per buffer. Mapped page-locked buffers let the GPU directly consume read destinations, eliminating an additional host-to-device copy. We also pin the process and its workers to a specific CPU core to mitigate the long I/O completion delays observed on our platform.

D Additional Evaluations

D.1 Calibration Set Sensitivity

We additionally assess the calibration-set sensitivity on Qwen3-4B-Instruct-2507 using the Pile (Gao et al., 2020) and C4 (Raffel et al., 2020) with 2-bit QuIP# quantization. All compensators are in 16-bit, and prefetching is disabled. As shown in Table 7, AquaPIC achieves similar gains over the uncompensated quantized model at both ranks on WikiText2 and C4 across calibration sets, suggesting limited sensitivity to the choice of calibration corpus.

Table 7: Perplexity evaluation using different calibration sets.

Config	**WikiText2** (↓)			**C4** (↓)		
Calibration Set	Pile	C4	RedPajama	Pile	C4	RedPajama
16-bit Original	10.06	10.06	10.06	17.20	17.20	17.20
2-bit Base	14.57	14.09	14.24	22.87	22.43	22.59
AquaPIC $R = 32$	12.61	12.52	12.51	20.29	19.99	20.23
AquaPIC $R = 64$	12.32	12.26	12.22	19.92	19.68	19.90

D.2 Effect of Quantizing Compensators

We evaluate the effect of compensator quantization on AquaPIC. For 8-bit, symmetric INT8 quantization is used. For 4-bit, GPTQ is applied to the optimized low-rank compensators.

Table 8 compares the perplexity of 16-bit, 8-bit, and 4-bit compensators on Llama-3.1-8B-Instruct and Qwen3-4B-Instruct-2507 with 2-bit GuidedQuant and QuIP# weights.

Table 8: Effect of compensator quantization on perplexity for Llama-3.1-8B-Instruct and Qwen3-4B-Instruct-2507 with 2-bit base models. The columns within each rank group indicate compensator precision.

Model	**Quant. method**	**Dataset**	**Original 16 bit**	**Base 2 bit**	AquaPIC ($R = 32$)			AquaPIC ($R = 64$)		
			Compensator Precision		16 bit	8 bit	4 bit	16 bit	8 bit	4 bit
Llama-3.1-8B-Instruct	Guided Quant	WikiText2 (↓)	7.22	40.01	26.76	26.75	27.12	22.11	22.11	22.54
		C4 (↓)	11.48	44.60	29.06	29.07	29.60	25.71	25.71	26.33
	QuIP#	WikiText2 (↓)	7.22	13.45	10.60	10.60	10.63	10.26	10.26	10.29
		C4 (↓)	11.48	22.09	17.55	17.56	17.57	17.00	17.01	17.04
Qwen3-4B-Instruct-2507	Guided Quant	WikiText2 (↓)	10.06	51.31	37.14	37.17	37.75	31.63	31.62	32.22
		C4 (↓)	17.20	42.53	28.63	28.64	28.93	26.12	26.12	26.45
	QuIP#	WikiText2 (↓)	10.06	14.24	12.48	12.48	12.47	12.21	12.20	12.21
		C4 (↓)	17.20	22.59	20.29	20.29	20.25	19.98	19.97	19.96

D.3 Effect of Prefetching

As AquaPIC uses the residual activations as inputs to an estimator, we evaluate the increase in perplexity from using residuals. Table 9 compares perplexity with prefetching on and off, using 16-bit compensators. With prefetching disabled, the block's immediate input is used as the input for the estimator. The results indicate that using the residual preserves most of the performance gains while enabling efficient prefetching.

Table 9: Effect of prefetching on perplexity on Llama-3.1-8B-Instruct. The numbers in parentheses indicate the delta between prefetching enabled and disabled.

Bits	Quantization Method	Prefetching	Wiki(↓)		C4(↓)	
		Rank	32	64	32	64
2	GuidedQuant	Disabled	26.66	21.90	28.70	25.32
		Enabled	26.76 (0.10)	22.11 (0.21)	29.06 (0.36)	25.71 (0.39)
	QuIP#	Disabled	10.31	10.03	16.77	16.35
		Enabled	10.60 (0.29)	10.26 (0.23)	17.55 (0.78)	17.00 (0.65)
3	GuidedQuant	Disabled	8.32	8.21	13.73	13.56
		Enabled	8.33 (0.01)	8.23 (0.02)	13.76 (0.03)	13.60 (0.04)
	QuIP#	Disabled	7.95	7.92	13.08	13.00
		Enabled	7.96 (0.01)	7.93 (0.01)	13.09 (0.01)	13.01 (0.01)

D.4 Different Cluster Counts

We evaluate the effect of the number of clusters per block, $C \in \{32, 64, 128\}$, using Qwen3-4B-Instruct-2507 quantized to 2 bits with QuIP# and compensation ranks $R \in \{32, 64\}$. Compensators are in 16-bit. Table 10 reports perplexity on WikiText2 and C4. All tested configurations improve over the uncompensated quantized model, and 128 clusters yield the lowest perplexity at each rank on both datasets; perplexity generally decreases as the number of clusters increases.

Table 10: Perplexity on Qwen3-4B-Instruct-2507 with different cluster counts. The columns within each rank group indicate the number of clusters C.

Dataset	Original 16 bit	QuIP# 2 bit	AquaPIC ($R = 32$)			AquaPIC ($R = 64$)		
			$C = 32$	$C = 64$	$C = 128$	$C = 32$	$C = 64$	$C = 128$
WikiText2 (↓)	10.06	14.24	12.62	12.64	12.51	12.41	12.32	12.22
C4 (↓)	17.20	22.59	20.43	20.36	20.23	20.10	19.98	19.90

E Clustering Algorithm of AquaPIC

Algorithm 1 describes the clustering procedure for one Attention/FFN block, with shared token assignments across its linear layers $\mathbb{L}$. The N columns of $\mathbf{Z}$ contain the block inputs used for initialization, while $\mathbf{X}_\ell = [\boldsymbol{x}_{\ell,1}, \ldots, \boldsymbol{x}_{\ell,N}]$ contains the corresponding inputs to linear layer ℓ. Partition 0 uses a fixed fallback compensator fitted to all calibration inputs; only partitions $1, \ldots, C-1$ are updated during clustering.

Algorithm 1 Clustering Algorithm of AquaPIC

Input: Block inputs $\mathbf{Z}$, layer inputs and weights $\{\mathbf{X}_\ell, \mathbf{W}_\ell, \hat{\mathbf{W}}_\ell\}_{\ell \in \mathbb{L}}$, number of partitions C, rank R; damping $\lambda > 0$; maximum iterations $I_{\max}$; stopping threshold τ
Output: Assignments $\boldsymbol{z}$; compensators $\Theta = \{(\mathbf{A}_{\ell,c}, \mathbf{B}_{\ell,c})\}_{\ell \in \mathbb{L},\, 1 \le c \le C}$

// Hessian-Weighted Low-Rank Compensator Fitting
function FIT($\Delta\mathbf{W}, \mathbf{X}, R, \lambda$)
  $\mathbf{H} \leftarrow \mathbf{X}\mathbf{X}^\top$
  $\mathbf{L} \leftarrow$ CHOLESKY($\mathbf{H} + \lambda\mathbf{I}$)
  $(\mathbf{U}_R, \mathbf{\Sigma}_R, \mathbf{V}_R^\top) \leftarrow$ RANDOMIZEDSVD($\Delta\mathbf{W}\mathbf{L}, R$)
  $\mathbf{B} \leftarrow \mathbf{U}_R\sqrt{\mathbf{\Sigma}_R}$; $\mathbf{A} \leftarrow \sqrt{\mathbf{\Sigma}_R}\mathbf{V}_R^\top\mathbf{L}^{-1}$
  return $(\mathbf{A}, \mathbf{B})$

// Pre-clustering and Compensator Initialization
Sample $\mathbf{P} \in \mathbb{R}^{K \times p}$ where $P_{i,j} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0,1)/\sqrt{K}$
$\boldsymbol{z} \leftarrow$ K-MEANS($\mathbf{P}^\top\mathbf{Z}, C-1$)
for $\ell \in \mathbb{L}$ **do**
  $\Delta\mathbf{W}_\ell \leftarrow \mathbf{W}_\ell - \hat{\mathbf{W}}_\ell$
  $d_\ell \leftarrow \|\mathbf{W}_\ell\mathbf{X}_\ell\|_F^2$
  $(\mathbf{A}_{\ell,0}, \mathbf{B}_{\ell,0}) \leftarrow$ FIT($\Delta\mathbf{W}_\ell, \mathbf{X}_\ell, R, \lambda$)
  for $c \in \{1, \dots, C-1\}$ **do**
    $\mathbf{X}_{\ell,c} \leftarrow$ Stack($\boldsymbol{x}_{\ell,i}$ for $i \in \{i \mid z_i = c\}$)
    $(\mathbf{A}_{\ell,c}, \mathbf{B}_{\ell,c}) \leftarrow$ FIT($\Delta\mathbf{W}_\ell, \mathbf{X}_{\ell,c}, R, \lambda$)

// Output-Error-Based Assignment and Compensator Updates
for $t = 1, \dots, I_{\max}$ **do**
  $\boldsymbol{z}^{\text{old}} \leftarrow \boldsymbol{z}$
  $e_{i,c} \leftarrow \sum_{\ell \in \mathbb{L}} \dfrac{\|(\Delta\mathbf{W}_\ell - \mathbf{B}_{\ell,c}\mathbf{A}_{\ell,c})\boldsymbol{x}_{\ell,i}\|_2^2}{d_\ell}$ for all i, c
  $z_i \leftarrow \arg\min_{c \in \{0,\dots,C-1\}} e_{i,c}$ for $i = 1, \dots, N$
  for $c \in \{1, \dots, C-1\}$ **do**
    $\mathbf{X}_{\ell,c} \leftarrow$ Stack($\boldsymbol{x}_{\ell,i}$ for $i \in \{i \mid z_i = c\}$)
    $(\mathbf{A}_{\ell,c}, \mathbf{B}_{\ell,c}) \leftarrow$ FIT($\Delta\mathbf{W}_\ell, \mathbf{X}_{\ell,c}, R, \lambda$)
  $\rho \leftarrow |\{i \mid z_i \neq z_i^{\text{old}}\}|/N$
  if $\rho < \tau$ **then**
    break
return $\boldsymbol{z}, \Theta$