# Any-Precision LLM: Low-Cost Deployment of Multiple, Different-Sized LLMs

**Yeonhong Park** [1] **Jake Hyun** [1] **SangLyul Cho** [1] **Bonggeun Sim** [1] **Jae W. Lee** [1]

## Abstract

Recently, considerable efforts have been directed towards compressing Large Language Models (LLMs), which showcase groundbreaking capabilities across diverse applications but entail significant deployment costs due to their large sizes. Meanwhile, much less attention has been given to mitigating the costs associated with deploying multiple LLMs of varying sizes despite its practical significance. Thus, this paper introduces *any-precision LLM*, extending the concept of any-precision DNN to LLMs. Addressing challenges in any-precision LLM, we propose a lightweight method for any-precision quantization of LLMs, leveraging a post-training quantization framework, and develop a specialized software engine for its efficient serving. As a result, our solution significantly reduces the high costs of deploying multiple, different-sized LLMs by overlaying LLMs quantized to varying bit-widths, such as 3, 4, ..., $n$ bits, into a memory footprint comparable to a single $n$-bit LLM. All the supported LLMs with varying bit-widths demonstrate state-of-the-art model quality and inference throughput, proving itself to be a compelling option for deployment of multiple, different-sized LLMs. The code is available at `https://github.com/SNU-ARC/any-precision-llm`.

## 1. Introduction

With the revolutionary success of Large Language Models (LLMs) across various applications, there have been many recent efforts to reduce the costs of their deployment. Specifically, there has been a considerable focus on compressing LLMs using techniques like pruning (Frantar & Alistarh, 2023; Zhang et al., 2023; Santacroce et al., 2023; Ma et al., 2023) or quantization (Lin et al., 2023; Frantar et al., 2023; Kim et al., 2023b), as the parameter size is the primary obstacle for their efficient deployment.

Meanwhile, there has been limited discussion on mitigating the costs associated with deploying multiple LLMs of varying sizes, despite its practical significance. Real-world scenarios often demand the dynamic adaptation of multiple LLMs, each with distinct model quality/inference latency trade-offs. This approach enables the effective handling of queries with varied latency constraints, enhancing the user experience. Moreover, it supports a popular generation acceleration technique: speculative decoding (Leviathan et al., 2023; Chen et al., 2023; Kim et al., 2023d). Despite these benefits, deploying multiple LLMs of varying sizes presents challenges. First, it exacerbates the already high memory costs of LLM deployment, and second, it necessitates training of multiple model versions when models of desired sizes are not readily available as open-source.

Any-precision LLM, an extension of the concept of any-precision DNN (Yu et al., 2021) to LLM, is a promising solution for the low-cost deployment of multiple, different-sized LLMs. Any-precision DNN refers to an $n$-bit quantized model capable of generating lower bit quantized models (($n$-1)-bit, ($n$-2)-bit, ...) simply by taking its most significant bits (MSBs). Applying this concept to LLM enables the utilization of multiple LLMs with varying sizes by storing only a single large LLM ($n$-bit model) in memory, while avoiding the additional overhead of training multiple LLMs.

Meanwhile, two challenges need to be resolved for effective implementation of any-precision LLM. First, a practical method for any-precision quantization of LLM is needed. The existing any-precision quantization method on DNN requires training the model from scratch, limiting its applicability to LLMs. Second, a new GPU kernel for quantized matrix-vector multiplication is required, which will translate the use of reduced bit-widths in any-precision LLMs into shorter inference times. Existing kernels for quantized matrix-vector multiplication are unable to load just a portion of each quantized weight value's bit-vector. Consequently, with existing kernels, opting for a model with a lower bit-width does not reduce memory bandwidth usage.

Thus, we in this paper make a strong case for any-precision LLM by addressing the two aforementioned issues. First, we build a lightweight method for any-precision quantiza-

[1]Seoul National University. Correspondence to: Jae W. Lee <jaewlee@snu.ac.kr>.

*Proceedings of the 41st International Conference on Machine Learning*, Vienna, Austria. PMLR 235, 2024. 

tion of LLM. Utilizing a post-training quantization (PTQ) framework, it first generates a low-bit model and then incrementally upscales it to higher bit-widths, conserving the any-precision property. Second, we develop a new software engine specialized for any-precision support, that effectively saves memory bandwidth for serving any-precision LLMs by changing the memory layout of weights. Our extensive experimental studies demonstrate that our solution is a powerful approach for the deployment of multiple, different-sized LLMs, achieving the following results:

- Our solution efficiently packs LLMs quantized to varying bit-widths, such as 3, 4, ... up to $n$ bits, into a memory footprint comparable to a single $n$-bit LLM.
- Our solution yields a set of quantized LLMs of varying bit-widths that, while offering any-precision support, match the quality of the state-of-the-art quantization techniques at each bit-width.
- Our solution, despite having to adopt a bit-interleaved (bitplane) memory layout for the support of any-precision, showcases high inference throughput, matching or even outperforming that of state-of-the-art quantized matrix-vector multiplication engines that do not support any-precision (Kim et al., 2023b).

## 2. Background

### 2.1. GPU Basics

**GPU Architecture Fundamentals.** A GPU, the de facto standard platform for executing LLMs, comprises a large number of processing elements called Streaming Multiprocessors (SMs). GPUs often include multi-level on-chip SRAM caches. A part of the L1 cache can be configured as shared memory, providing a memory space that can be directly controlled by programmers.

**Execution Model.** GPUs use a large number of threads to execute operations, which are known as kernels. Threads are structured into thread blocks whose execution is scheduled on SMs. All threads within a thread block share the same shared memory space. Within each thread block, threads are further organized into a set of warps, with each warp consisting of 32 consecutive threads. All threads within a warp execute the same instruction at the same time.

### 2.2. LLM Quantization

This section discusses recent advancements in LLM quantization with a specific emphasis on weight-only, post-training quantization (PTQ). In LLMs, there is a pronounced shift towards quantizing only the weights, as the dominant bottleneck in inference throughput is the memory constraint imposed by the size of weight parameters, rather than computational requirements. Moreover, Post-Training Quantization (PTQ) has become a favored method for quantizing LLMs due to its practicality. Although Quantization-Aware Training (QAT) typically yields superior performance, its high training expense frequently renders it impractical (Liu et al., 2023; Dettmers et al., 2023; Kim et al., 2023a).

GPTQ (Frantar et al., 2023), a pioneering work on weight-only PTQ for LLM, formulates quantization as a layer-wise weight reconstruction problem. GPTQ methodically quantizes each channel in iterations, simultaneously adjusting the remaining not-yet-quantized weights to correct for quantization-induced errors. AWQ (Lin et al., 2023) performs per-channel scaling to safeguard a small fraction of salient weights as a preprocessing step. Similarly, QuIP (Chee et al., 2023) preprocesses weights to be more amenable to quantization, yielding impressive results even at 2-bit precision. However, its practical utility is in question as it adds a substantial runtime overhead.

While the aforementioned methods employ uniform quantization, non-uniform quantization may be a more effective alternative as it better captures the weight distributions (Gholami et al., 2021). SqueezeLLM (Kim et al., 2023b) proposes a clustering-based LLM quantization that considers the sensitivity of each weight. Somewhat orthogonal to the aforementioned studies that primarily focus on rounding schemes, there are also proposals to use mixed precision as a way of allocating more bits to sensitive weights (Kim et al., 2023b; Dettmers et al., 2024; Lee et al., 2024).

## 3. Motivation

### 3.1. Need for Deploying Multiple, Different-sized LLMs

Deploying a set of different-sized LLMs provides significant practical advantages. It enhances user experience by effectively handling queries with varying latency requirements. Depending on the specific needs of users and applications, some queries require quick responses, while others can tolerate slower response times. Latency requirements become even more diverse when serving multiple different tasks concurrently, a common use case of LLMs. For instance, queries for interactive tasks like chatbots are mostly latency-sensitive, while tasks like document analysis, often handled in the background, allow for more relaxed response times. In fact, the approach of adaptively selecting different DNN models with various accuracy-latency trade-offs has been widely studied as a way to effectively meet user requirements due to its practical significance (Wan et al., 2020; Zhang et al., 2020; Han et al., 2016).

Another scenario that necessitates multiple LLMs of varying sizes is speculative decoding. This popular technique boosts the throughput of a large model by additionally utilizing one or more smaller draft models (Leviathan et al., 2023; Chen et al., 2023; Kim et al., 2023d; Miao et al., 2023).

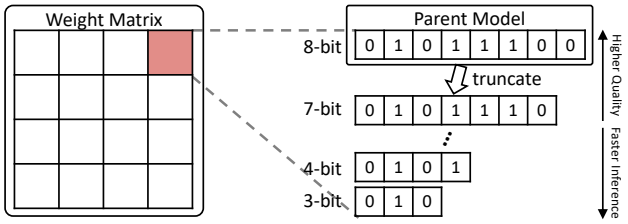


*Figure 1.* Concept of any-precision quantization.

*Table 1.* Memory savings of any-precision LLM when deployed on various sets of bit-widths for Llama-2-7B.

| Supported Bit-widths | Any-Precision LLM | Separate Deployment | Memory Savings |
|---|---|---|---|
| {3,6} | 5.6 GB | 8.3 GB | 1.49× |
| {4,8} | 7.7 GB | 10.8 GB | 1.40× |
| {3,4,6} | 5.6 GB | 12.1 GB | 2.15× |
| {3,4,8} | 7.7 GB | 13.7 GB | 1.76× |
| {3,4,6,8} | 7.9 GB | 19.1 GB | 2.41× |
| {3,4,5,6,7,8} | 8.4 GB | 29.9 GB | 3.56× |

### 3.2. Challenges of Deploying Multiple, Different-sized LLMs

In deploying multiple LLMs, we face two practical challenges: memory overhead and training costs.

**Challenge 1: Memory Overhead.** First, a large memory capacity overhead is introduced. As LLMs are typically substantial in size, maintaining even a few additional smaller models incurs significant costs. For example, deploying three tiers of LLMs with varying sizes — a large base model, a half-sized model, and a quater-sized model — nearly doubles on the total memory requirement.

**Challenge 2: Training Costs.** Acquiring multiple different-sized LLMs is challenging in itself. While some open-source LLMs such as OPT (Zhang et al., 2022) offer a comprehensive range of models with varying parameter counts, this is usually not the case. Most open-source LLMs offer only one to three variants, limiting their versatility across different use scenarios. If a model in the desired size is unavailable, users must create it by themselves. However, training an LLM is very costly due to its high computational needs and large corpus requirement. One approach is to distill the available large model into smaller ones instead of training smaller models from scratch (Hinton et al., 2015; Hsieh et al., 2023; Gu et al., 2024; Zhao et al., 2022; Agarwal et al., 2024). While this approach reduces computation costs, it still entails non-trivial engineering challenges, including the need to assemble a proper set of training data and finding favorable hyperparameters for training.

These two challenges become particularly prominent when individuals run LLMs on personal platforms like desktops and mobile devices. In these scenarios — unlike with datacenter-scale inference — compute, memory, and even engineering resources are severely limited. Thus, this paper focuses on addressing the challenges associated with deploying multiple, different-sized LLMs for on-device inference.

### 3.3. Our Solution: Any-Precision LLM

**Concept.** Any-precision quantization, initially introduced in prior work (Yu et al., 2021), is a promising solution to mitigate the costs of deploying multiple different-sized LLMs. Figure 1 visualizes the concept of any-precision quantization. The core is to derive smaller models (7-bit, 6-bit quantized model, ...) from a large model (8-bit quantized model), referred to as the parent model, by taking only the upper bits of its parameters. Of course, a special method tailored for any-precision quantization is required to ensure that taking just the prefixes of the parent model parameters does not result in significant quality drops. The any-precision approach is highly memory-efficient as it allows the utilization of varying bit-width models by only storing in memory 1) the quantized weights of the parent model and 2) a set of quantization parameters (e.g. centroid values) associated with each supported bit-width, relatively small in size. Also, there is no need to train multiple models.

Table 1 gives a quantitative analysis on the memory savings of any-precision LLM, the application of any-precision quantization to LLMs. Without any-precision quantization, adaptively using models of different bit-widths requires the deployment of separate models, each taking up its own memory space. We refer to this strategy as separate deployment. We compare the required memory space of separate deployment against any-precision LLM, for various scenarios requiring different sets of bit-widths with the Llama-2-7B model. Any-precision LLM significantly reduces memory costs across a range of scenarios, achieving a maximum saving of 3.56× when supporting all bit-widths from 3 to 8.

**Challenges of Any-Precision LLM.** While the concept itself is appealing, there are critical issues in directly applying the method of the original work (Yu et al., 2021), which mainly targets CNN models, to LLMs. First, it is a quantization-aware training (QAT) scheme, requiring models to be trained from scratch. During training, the forward pass quantizes parameters to varying bit-widths so that the resulting model is robust to any-precision quantization. However, for LLMs, training is not affordable to most users. Second, this work has no regard for memory bandwidth saving. The entire $n$-bit parameters of the parent model are loaded into memory, only then being further quantized into lower bit-width weights by bit-shifting as needed. This strategy makes sense for CNN models, which are usually compute-bound. On the other hand, on-device LLM in-

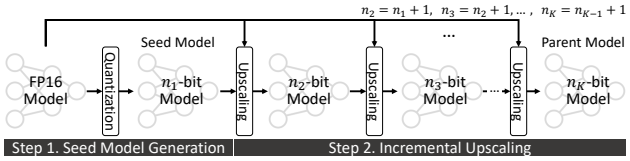


*Figure 2.* Any-precision quantization via incremental upscaling.

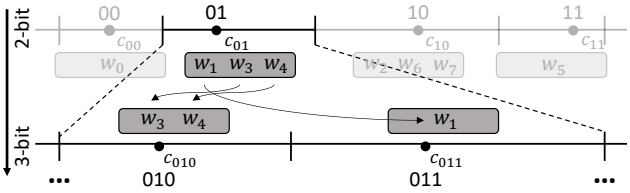


*Figure 3.* Incremental upscaling from 2 to 3-bit on non-uniform quantization methods.

ference is highly memory-bound due to its low arithmetic intensity. Consequently, the memory load of weight parameters is the single primary performance bottleneck (Kim et al., 2023c). Hence, when the original method is applied to LLMs as-is, operating at low bit-widths may provide little inference latency improvements over that of the parent model. A new solution is required for LLMs, one that incorporates both a low-cost any-precision quantization method and a specialized software engine wherein reduced precision inference directly translates to actual speed-ups.

## 4. Any-Precision Quantization for LLM

### 4.1. Incremental Upscaling

We propose a novel approach to any-precision quantization of LLMs, termed *incremental upscaling*. Figure 2 illustrates the two-stage flow of the any-precision quantization, utilizing incremental upscaling. Assuming a list of candidate bit-widths ($\{n_k\}_{k=1}^{K}$), the initial step quantizes the model to the minimum supported bit-width ($n_1$), which we refer to as *seed model*. Subsequently, we incrementally upscale the seed model one bit at a time, until we obtain the final $n_K$-bit parent model. For every incremental upscale from an $n_i$-bit model to an $n_{i+1}$-bit model, all parameters of the $n_i$-bit model are inherited to the $n_{i+1}$-bit model, and a single additional bit is appended to the end of each parameter.

### 4.2. Non-uniform Quantization-based Incremental Upscaling

For our incremental upscaling-based approach of any-precision quantization, a particular quantization method must be adopted as a backbone for both the seed model generation and the subsequent upscaling process. While any PTQ method can be employed for this purpose, we rule out the use of QAT method to ensure that our solution does not involve an expensive training process. Among various PTQ methods, an optimal choice would be one that 1) demonstrates state-of-the-art quantization results for the generation of a quality seed model and 2) seamlessly extends to the incremental upscaling process. To this end, we utilize the state-of-the-art clustering-based non-uniform quantization scheme, SqueezeLLM (Kim et al., 2023b), as the backbone method. It delivers state-of-the-art results and is readily compatible with incremental upscaling.

*Table 2.* Perplexity increase observed in uniform quantization methods (AWQ, GPTQ) with the application of incremental upscaling (IU). The evaluation is performed using Llama-2-7B on Wikitext2.

| | 3-bit | 4-bit | 5-bit | 6-bit | 7-bit | 8-bit |
|---|---|---|---|---|---|---|
| AWQ | 6.24 | 5.59 | 5.50 | 5.48 | 5.47 | 5.47 |
| AWQ+IU | | INF | 22.50 | 8.16 | 6.07 | 5.67 |
| GPTQ | 6.73 | 5.68 | 5.52 | 5.48 | 5.47 | 5.47 |
| GPTQ+IU | | 12.7 | 6.00 | 5.79 | 5.74 | 5.73 |

Incremental upscaling with the clustering-based quantization method is straightforward. In SqueezeLLM, quantization is a process of clustering the parameters, where values within each cluster are rounded to its centroid. In this light, incremental upscaling can be achieved by further dividing each cluster into two sub-clusters. Specifically, we perform a weighted K-means clustering on the values of each cluster to generate the two sub-clusters. A sensitivity metric based on an approximated second-order derivative is used during clustering, like in the original method.

Figure 3 visualizes the upscaling process on the clustering-based non-uniform quantization method, assuming the case of producing a 3-bit model from a 2-bit model. In this instance, three weight parameters ($w_1, w_3, w_4$) initially assigned to cluster `01` — previously rounded to its centroid $c_{01}$ — are now divided into two new sub-clusters, namely `010` and `011`, each with its respective centroids $c_{010}$ and $c_{011}$. The same process is applied to the remaining three clusters (`00`, `10`, `11`), resulting in a final total of eight sub-clusters.

In contrast to clustering-based methods, state-of-the-art *uniform* quantization methods often involve complicated mechanisms such as weight reconstruction (Frantar et al., 2023) and per-channel scaling (Lin et al., 2023) that are not readily compatible with incremental upscaling. For example, Table 2 presents the results of applying incremental upscaling to two state-of-the-art uniform methods (AWQ, GPTQ), using a 3-bit model as the seed model. The upscaled models consistently underperform the independently quantized models, showing unacceptable degradation at low bit-widths such as 4. We provide an in-depth discussion on the challenges of applying incremental upscaling on existing uniform quantization methods, along with additional experimental results, in Appendix E.

## 5. Specialized Software Engine

### 5.1. Need for New Software Engine

There are multiple GPU kernels designed for the efficient execution of weight-only-quantized LLMs (NVIDIA; Lin et al., 2023; Frantar et al., 2023; Kim et al., 2023b; turboderp; Park et al., 2024). While demonstrating promising performance, most of these existing implementations cannot support any-precision quantization because of their way of representing quantized weights. They use a *bitpacking*-based representation, illustrated in Figure 4-(a). Bitpacking-based representations store quantized weights sequentially in a single 1-D array. With this representation, the entire weight array has to be loaded even when running a model in a reduced bit-width. For example in Figure 4-(a), even when executing a 2-bit or 3-bit model, the full 4-bit values have to be read from memory. This is because of the coarse-grained memory access granularity of GPUs, typically 128 bytes. This results in virtually no performance improvements.

On the contrary, *bitplane*-based representation is a suitable choice for any-precision LLM support. Bitplane-based representation decomposes quantized weights into $n$ bit-vectors, where $n$ is the bit-width. Each bit-vector is formed by taking each bit position of the quantized values. Figure 4-(b) illustrates the bitplane-based representation. In this representation, any runtime request of reduced bit-width directly translates into proportional speedup, as we can simply load the specified amount of bits. While relatively common in CPU GEMM implementations (Cowan et al., 2020; Umuroglu & Jahre, 2017), its adoption in GPUs has not gained much attention yet. Concurrent to our work, LUT-GEMM proposes a quantized matrix-vector multiplication kernel adopting a kind of bitplane-based weight representation (Park et al., 2024), but it lacks support for any-precision as it necessitates the generation of distinct weight layouts to accommodate different bit-widths. More importantly, LUT-GEMM strictly requires weights to adhere to a specific format called BCQ (binary-coding quantization) — BCQ cannot support the codebook-based non-uniform quantization method, which we identify as optimal for any-precision LLM. Hence, a novel software engine that fully supports both bitplane-based weight representation and non-uniform quantization is needed.

### 5.2. System Overview

Figure 5 is an overview of our engine for any-precision LLM. This figure assumes a scenario where support for 3, 4 and 8-bit is available, and the engine is requested to run a 4-bit model. For simplicity, we show only a single linear operation. Weight matrix bitplanes of bit 0 to bit 7, along with tables containing centroid values for the supported bit-widths (3, 4, 8-bit), are stored in memory. Assuming channel-wise quantization, the centroid tables have rows

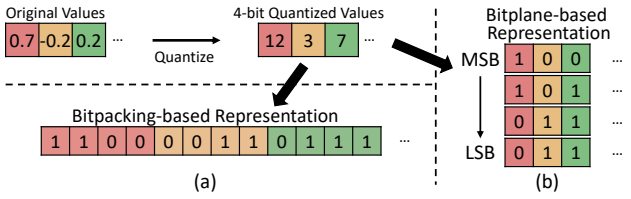


*Figure 4.* Comparison of (a) bitpacking-based and (b) bitplane-based representations of quantized weights.

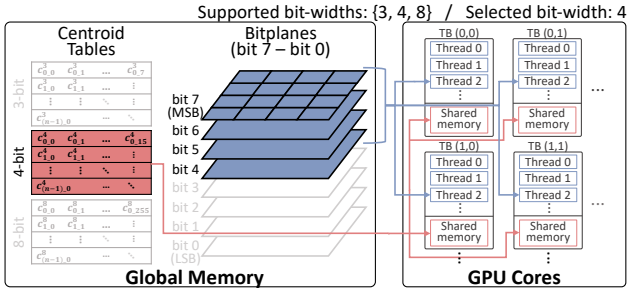


*Figure 5.* Overview of our specialized engine for any-precision LLM. We illustrate the engine running a 4-bit model while support for 3, 4 and 8-bit is available.

equal to the number of output channels ($n$), with each row containing $2^k$ values, where $k$ refers to the bit-width. For each operation, only necessary data is loaded from memory, which in this case includes the centroid table for 4-bit and the bitplanes for bit 4, 5, 6, and 7 in this example. The rows of the centroid table are scattered across the shared memory of different thread blocks since they are frequently accessed by all the threads in the block. Meanwhile, the threads load non-overlapping regions of the weight bitplanes.

**Thread-Level Operations.** Figure 6 depicts the five step thread-level operation, assuming a bit-width of 4. ❶ First, each thread starts by loading 32 input activation values along with their corresponding weights, which are four 32-bit bit-vectors. ❷ Next, the four bit-vectors are rearranged so that the bits of each weight align contiguously. This operation is equivalent to the bit-transpose of eight 4-by-4 bit matrices. ❸ Subsequently, the bit-transposed bit-vectors are shifted and masked to obtain the indices for centroid lookup. ❹ The dequantization process is then completed by fetching centroids from the table in the shared memory. ❺ As the final step, Multiply-Accumulate (MAC) operations are conducted on the dequantized weights with the input activations. Threads iteratively execute these five steps until they collectively complete processing the channels assigned to them, alongside other threads in the warp.

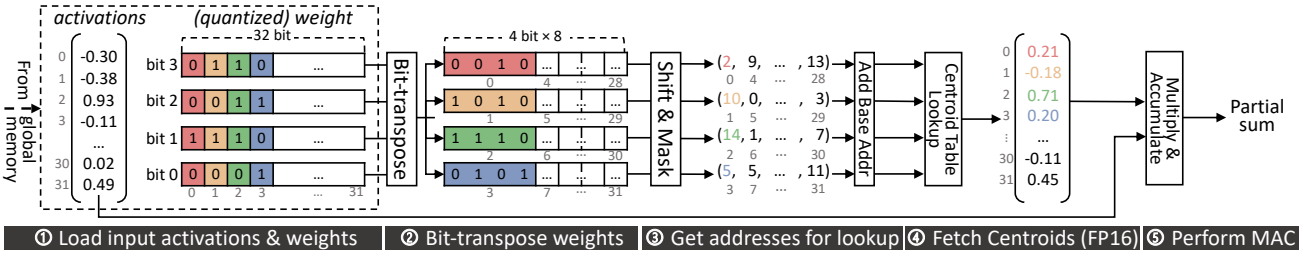


*Figure 6.* Five steps of thread-level operations.

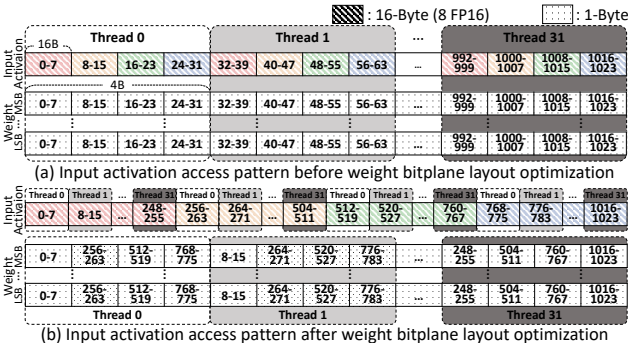


*Figure 7.* Memory access pattern of the 32 threads in a warp (a) before and (b) after the weight bitplane layout optimization. Simultaneously accessed input activation blocks are in the same colors.

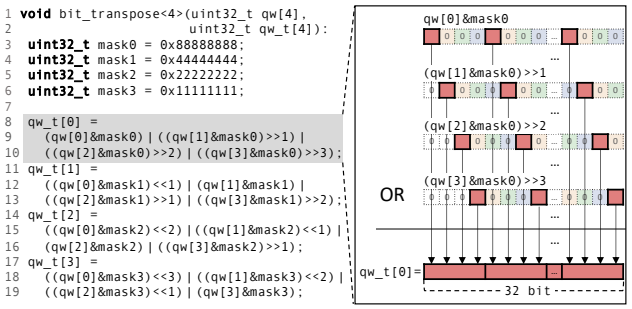


*Figure 8.* Efficient bit-transpose operation of our engine.

### 5.3. GPU Kernel Optimization

This section describes three GPU kernel optimization techniques aimed at addressing inefficiencies stemming from the characteristics of bitplane-based quantized GEMM.

**Weight Bitplane Layout Optimization.** The bitplane-based representation causes a unique problem for input activation loading. Figure 7-(a) illustrates this issue. When reading from GPU memory, it is preferable for all 32 threads in a warp to access consecutive memory locations so that the memory accesses can be coalesced into a single request. As memory access granularity is 128 bytes, it is optimal when each one of the 32 threads in a warp load consecutive 4-byte blocks (32 weights) from each weight bitplane, to a total of 128 bytes. Furthermore, threads should also load the corresponding consecutive 32 input activation values. Since each activation value is in FP16, each thread needs to load a 64-byte (32×2) block. As CUDA limits the per-thread maximum load size to 16 bytes, four memory accesses are necessary. At each of the four memory accesses, the 32 threads access non-contiguous locations in memory — for the first access, this is activations 0-7 for thread 0, activations 32-39 for thread 1, and so on — which is not ideal.

Hence, we suggest permuting bytes in the bitplanes to ensure that threads access activations in a coalesced manner. Figure 7-(b) visualizes the transformation in the weight bitplane layout and the resulting activation access pattern. The indices of weights accessed by each thread are no longer sequential; instead, the indices of weights in each byte of a 4-byte block are now 256 units apart . For instance, thread 0 processes weights 0-7, 256-263, 512-519, and 768-775, as opposed to weights 0-31. Consequently, each memory access of the 32 threads for the activations leads to a coalesced memory access pattern. Such byte permutation of bitplanes is performed as a part of pre-processing.

**Efficient Bit-transpose.** In order to dequantize weights, we have to preform a bit-transpose step (step 2 in Figure 6) as the weights are represented as bitplanes. This step requires a large number of bitwise operations, becoming the main overhead. Even an optimized bit-transpose algorithm requires 38 bitwise operations just to process an 8×8 bit matrix (Warren, 2012).

Figure 8 describes our optimized bit-transpose algorithm. The example in the figure assumes a bit-width of 4, taking four 32-bit bit-vectors as input (`qw[4]`) and outputting four new 32-bit bit-vectors (`qw_t[4]`) in which the bits of weights are arranged contiguously. The key is to treat a 32-bit bit-vector as eight 4-bit sub-bit-vectors, ensuring that operations on them effectively function like SIMD (Single Instruction Multiple Data) operations. With this approach,

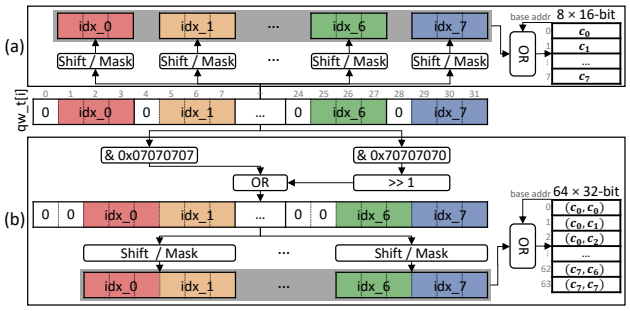


*Figure 9.* Operations to obtain centroid table lookup addresses (a) before and (b) after table lookup merging.

we can pack bit positions 0, 4, ..., and 28 from the four input bit-vectors into a single 32-bit bit-vector with 10 bitwise operations. This process is repeated three more times for the other bit positions, completing the entire task. This totals to 40 bitwise operations — nearly half the number required (76) if you were to apply the 8×8 bit matrix transpose algorithm twice instead. This approach is also applicable for bit-widths of 2 and 8-bit by interpreting a 32-bit bit-vector as sixteen 2-bit and four 8-bit sub-bit-vectors, respectively. For non-power-of-2 bit-widths, we seamlessly apply the algorithm for the next larger bit-width that is a power of 2.

**Merging Table Lookups.** Bitwise operations for the centroid table index calculation (Step 3 in Figure 6) can also become a bottleneck, particularly at small bit-widths. Figure 9-(a) shows how this step works in detail, assuming a bit-width of 3. After Step 2 of Figure 6, we obtain four 32-bit bit-vectors (`qw_t[0]`, ..., `qw_t[3]`), each containing eight 3-bit indices (`idx_0`, ..., `idx_7`) zero-extended to 4 bits. To complete the dequantization for each bit-vector, eight centroid table lookups are necessary, and each lookup entails three bitwise operations: the first two for the shifting and masking to derive indices, and the third for adding the base address of the table to the indices. Thus, 24 bitwise operations are required for each bit-vector.

To alleviate this overhead on the 3-bit case, we halve the number of table lookups by merging two lookups into one, as depicted in Figure 9-(b). Two adjacent 3-bit indices ([`idx_0`, `idx_1`], [`idx_2`, `idx_3`], ...) are merged into a single 6-bit value, serving as a new index for the lookup. The table is expanded to contain 64 entries instead of 8, containing all possible pairs of centroids ($[c_0, c_0]$, $[c_0, c_1]$, ...). Now we retrieve two centroid values with a single lookup. The number of required bitwise operations is reduced to 16 for each 32-bit bit-vector: 12 for shifting, masking, and adding the base address, and the remaining 4 for rearranging the 32-bit bit-vector. By allowing an acceptable increase in shared memory usage, we mitigate the compute costs.

# 6. Evaluation

Through extensive experiments, we demonstrate that our proposal is an effective way of deploying multiple, different-sized LLMs by proving the following two arguments:

- A set of quantized models generated from incremental upscaling, when integrated with the non-unifrom quantization method, match the state-of-the-art quantization results at their respective bit-widths. (Section 6.1)
- Our specialized engine matches or even outperforms existing engines while providing memory-efficient any-precision support, a feature lacking in the existing engines. (Section 6.2 & Section 6.3)

## 6.1. Any-Precision Quantization Results

**Methodology.** We evaluate 4 to 8-bit models obtained through incremental upscaling, using a 3-bit SqueezeLLM model as the seed model. The results are compared with the 4 to 8-bit SqueezeLLM models. We benchmark our method on LLaMA-2-7B (Touvron et al., 2023), Mistral-7B (Jiang et al., 2023), and three OPT models (6.7B, 2.7B, 1.3B) (Zhang et al., 2022). We evaluate the models with two metrics: perplexity on three datasets (WikiText2 (Merity et al., 2016), PTB (Marcus et al., 1994), C4 (Raffel et al., 2023)) and zero-shot accuracy on five tasks (ARC-easy/challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), PIQA (Tata & Patel, 2003), WinoGrande (Sakaguchi et al., 2021)). Experimental details are in Appendix A.

**Results.** Table 3 shows the perplexity results. The upscaled models (SqLLM+IU) nearly match the independently quantized models (SqLLM) across various models, datasets, and bit-widths. Except in two cases (OPT-6.7B, OPT-1.3B 4-bit on PTB), the increase in perplexity is negligible (< 0.1).

A similar trend is observed in the zero-shot task results. Table 4 presents the average accuracy across five zero-shot tasks. The upscaled models achieve the same level of accuracy as the independently quantized models. The accuracy drop from upscaling is within 0.2%, with upscaled models even outperforming in some cases. Full results for each task are provided in Appendix B.

These results demonstrate that with a single $n$-bit parent model generated through incremental upscaling, we can utilize the full range of 3 to $n$-bit models, all achieving state-of-the-art quality at their respective bit-widths.

**Runtime of Any-Precision Quantization.** Our any-precision quantization scheme is efficient as it does not require training and is highly parallelizable. We measure the runtime of the any-precision quantization process, beginning with a 3-bit seed model and progressing up to the final 8-bit parent model, on an Intel i9-13900K CPU with 24 cores. Table 5 shows the results. It takes less than a minute to complete the whole process, even for 7B-scale models.

*Table 3.* Perplexity on Wikitext2 (Wiki), C4 and Penn Treebank (PTB) for vanilla SqueezeLLM (SqLLM) and SqueezeLLM integrated with incremental upscaling (SqLLM+IU) using a 3-bit seed model. We also report the difference between the two methods, highlighting cases in red where the increase in perplexity exceeds 0.1.

| | | 3-bit | | | 4-bit | | | 5-bit | | | 6-bit | | | 7-bit | | | 8-bit | | | FP16 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Wiki | PTB | C4 | Wiki | PTB | C4 | Wiki | PTB | C4 | Wiki | PTB | C4 | Wiki | PTB | C4 | Wiki | PTB | C4 | Wiki | PTB | C4 |
| **Llama-2-7B** | SqLLM | 6.13 | 8.95 | 8.20 | 5.61 | 8.24 | 7.44 | 5.50 | 8.13 | 7.30 | 5.47 | 8.11 | 7.27 | 5.47 | 8.10 | 7.27 | 5.47 | 8.10 | 7.26 | 5.47 | 8.10 | 7.26 |
| | SqLLM+IU | - | - | - | 5.62 | 8.23 | 7.45 | 5.50 | 8.12 | 7.30 | 5.47 | 8.10 | 7.27 | 5.47 | 8.10 | 7.27 | 5.47 | 8.10 | 7.26 | | | |
| | Δ | - | - | - | ▲0.01 | ▼0.01 | ▲0.01 | - | ▼0.01 | - | - | ▼0.01 | - | - | - | - | - | - | - | | | |
| **Mistral-7B** | SqLLM | 5.94 | 9.61 | 9.33 | 5.36 | 8.65 | 8.55 | 5.27 | 8.52 | 8.41 | 5.25 | 8.47 | 8.39 | 5.25 | 8.46 | 8.38 | 5.25 | 8.46 | 8.38 | 5.25 | 8.46 | 8.38 |
| | SqLLM+IU | - | - | - | 5.39 | 8.68 | 8.57 | 5.28 | 8.52 | 8.42 | 5.25 | 8.48 | 8.39 | 5.25 | 8.46 | 8.38 | 5.24 | 8.46 | 8.38 | | | |
| | Δ | - | - | - | ▲0.03 | ▲0.03 | ▲0.02 | ▲0.01 | - | ▲0.01 | - | ▲0.01 | - | - | - | - | ▼0.01 | - | - | | | |
| **OPT-6.7B** | SqLLM | 11.60 | 13.35 | 13.83 | 10.96 | 12.59 | 13.17 | 10.84 | 12.55 | 13.07 | 10.84 | 12.57 | 13.05 | 10.86 | 12.52 | 13.05 | 10.86 | 12.52 | 13.05 | 10.86 | 12.52 | 13.05 |
| | SqLLM+IU | - | - | - | 11.01 | 12.82 | 13.20 | 10.88 | 12.55 | 13.07 | 10.84 | 12.55 | 13.05 | 10.84 | 12.55 | 13.05 | 10.86 | 12.55 | 13.05 | | | |
| | Δ | - | - | - | ▲0.05 | ▲0.23 | ▲0.03 | ▲0.04 | - | - | - | ▼0.02 | - | ▼0.02 | ▲0.03 | - | - | ▲0.03 | - | | | |
| **OPT-2.7B** | SqLLM | 13.97 | 15.86 | 16.11 | 12.70 | 14.75 | 14.95 | 12.42 | 14.47 | 14.78 | 12.47 | 14.41 | 14.75 | 12.45 | 14.41 | 14.73 | 12.47 | 14.44 | 14.73 | 12.47 | 14.44 | 14.73 |
| | SqLLM+IU | - | - | - | 12.72 | 14.78 | 15.02 | 12.47 | 14.52 | 14.78 | 12.45 | 14.47 | 14.75 | 12.47 | 14.44 | 14.75 | 12.47 | 14.44 | 14.73 | | | |
| | Δ | - | - | - | ▲0.02 | ▲0.03 | ▲0.07 | ▲0.05 | ▲0.05 | - | ▼0.02 | ▲0.06 | - | ▲0.02 | ▲0.03 | ▲0.02 | - | - | - | | | |
| **OPT-1.3B** | SqLLM | 16.30 | 18.39 | 18.39 | 14.93 | 16.45 | 16.81 | 14.64 | 16.20 | 16.62 | 14.61 | 16.20 | 16.55 | 14.61 | 16.20 | 16.55 | 14.61 | 16.17 | 16.55 | 14.64 | 16.17 | 16.55 |
| | SqLLM+IU | - | - | - | 14.95 | 16.59 | 16.88 | 14.66 | 16.23 | 16.62 | 14.64 | 16.20 | 16.55 | 14.64 | 16.20 | 16.55 | 14.64 | 16.17 | 16.55 | | | |
| | Δ | - | - | - | ▲0.02 | ▲0.14 | ▲0.07 | ▲0.02 | ▲0.03 | - | ▲0.03 | - | - | ▲0.03 | - | - | ▲0.03 | - | - | | | |

*Table 4.* Average zero-shot accuracy on five tasks — ARC-easy, ARC-challenge, HellaSwag, PIQA, and WinoGrande — for vanilla SqueezeLLM (SqLLM) and SqueezeLLM integrated with incremental upscaling (SqLLM+IU) using a 3-bit seed model. We also report the difference between the two methods.

| | | 3-bit | 4-bit | 5-bit | 6-bit | 7-bit | 8-bit | FP16 |
|---|---|---|---|---|---|---|---|---|
| Llama-2-7B | SqLLM | 66.2 | 68.3 | 68.6 | 68.8 | 68.9 | 68.9 | 69.0 |
| | SqLLM+IU | - | 68.2 | 68.8 | 68.9 | 68.9 | 69.0 | |
| | Δ | - | ▼0.1 | ▲0.2 | ▲0.1 | - | ▲0.1 | |
| Mistral-7B | SqLLM | 71.5 | 73.2 | 73.8 | 74.0 | 74.1 | 74.2 | 74.1 |
| | SqLLM+IU | - | 73.3 | 73.8 | 74.0 | 74.2 | 74.1 | |
| | Δ | - | ▲0.1 | - | - | ▲0.1 | ▼0.1 | |
| OPT-6.7B | SqLLM | 58.6 | 59.9 | 60.6 | 60.7 | 60.7 | 60.8 | 60.8 |
| | SqLLM+IU | - | 60.3 | 60.6 | 60.7 | 60.8 | 60.7 | |
| | Δ | - | ▲0.4 | - | - | ▲0.1 | ▼0.1 | |
| OPT-2.7B | SqLLM | 54.3 | 55.8 | 56.3 | 56.2 | 56.3 | 56.3 | 56.4 |
| | SqLLM+IU | - | 55.7 | 56.1 | 56.2 | 56.4 | 56.3 | |
| | Δ | - | ▼0.1 | ▼0.2 | - | ▲0.1 | - | |
| OPT-1.3B | SqLLM | 50.7 | 52.7 | 52.9 | 53.4 | 53.4 | 53.3 | 53.3 |
| | SqLLM+IU | - | 52.6 | 53.1 | 53.3 | 53.4 | 53.3 | |
| | Δ | - | ▼0.1 | ▲0.2 | ▼0.1 | - | - | |

*Table 5.* Runtime (sec) of the proposed any-precision quantization scheme, composed of two stages: seed model generation (Seed Gen) and incremental upscaling (IU).

| | Llama-2-7B | Mistral-7B | OPT-6.7B | OPT-2.7B | OPT-1.3B |
|---|---|---|---|---|---|
| Seed Gen | 36.2 | 37.2 | 37.0 | 14.0 | 6.4 |
| IU | 15.6 | 18.2 | 12.8 | 6.2 | 3.0 |
| Total | 51.8 | 55.4 | 49.8 | 20.2 | 9.4 |

### 6.2. Kernel Microbenchmarks

**Methodology.** We evaluate the latency of matrix-vector multiplication of our kernel on the three matrix dimensions used in Llama-2-7B, and compare with the existing kernel for non-uniform quantization proposed in SqueezeLLM (Kim et al., 2023b). Note that the SqueezeLLM kernel only supports 3 and 4 bits. We conduct experiments on three GPUs of varying scales: RTX 4090 (desktop), RTX 4070 Laptop (laptop), and Jetson AGX Orin 64 GB (mobile). We use NVIDIA Nsight Compute to measure latency.

**Results.** Table 6 shows the results. Our kernel consistently achieves low latency for various matrix sizes and platforms, showing a near-linear improvement against the cuBLAS FP16 baseline as the bit-width decreases. Compared to the SqueezeLLM kernel, our kernel performs on par on the RTX 4090 and RTX 4070 Laptop, and goes on to exhibit substantial improvements on Jetson. Note that our kernel achieves the performance despite adopting the bitplane-based weight representation to support any-precision, which entails non-trivial engineering challenges such as addressing the increased amount of bitwise operations for bit-transpose. Additional results with a wider range of matrix dimensions and comparison against uniform quantization kernels can be found in Appendix C.1 and C.2.

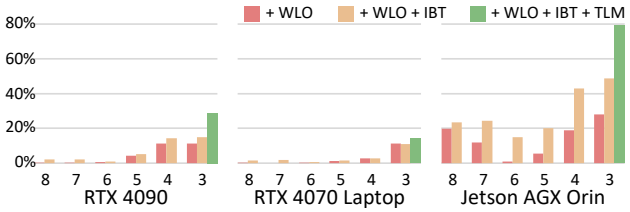


*Figure 10.* Ablation study of kernel optimization techniques. The speedup over the baseline for matrix-vector multiplication of dimensions (1,4096) and (11008,4096) is reported.

**Ablation Study.** Figure 10 shows how the kernel performance improves with the introduction of each of the three optimization techniques: 1) weight bitplane layout optimization (WLO); 2) improved bit-transpose algorithm over the existing one (Warren, 2012) (IBT); and 3) table lookup merging (TLM) for the case of 3-bit. While all three optimization techniques significantly contribute to improving kernel performance, there are two notable trends. First, the optimization effect is more pronounced on lower bit-widths. This is because, at higher bit-widths, the global memory bandwidth tends to be the sole bottleneck, whereas our techniques primarily optimize cache access patterns (WLO) and computations (IBT, TLM) — aspects that become more critical at lower bit-widths. Second, the optimization effect is more pronounced on Jetson AGX Orin. This is attributed to its limited on-chip cache and compute resources.

*Table 6.* Matrix-vector multiplication speedup of our kernel over the cuBLAS FP16 baseline on three weight sizes of Llama-2-7B, compared against the existing kernel for non-uniform quantization (SqLLM).

| | | (1, 4096) × (4096, 4096) | | | | | | (1, 4096) × (11008, 4096) | | | | | | (1, 11008) × (4096, 11008) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 3-bit | 4-bit | 5-bit | 6-bit | 7-bit | 8-bit | 3-bit | 4-bit | 5-bit | 6-bit | 7-bit | 8-bit | 3-bit | 4-bit | 5-bit | 6-bit | 7-bit | 8-bit |
| **RTX 4090** | Ours | ×3.99 | ×3.03 | ×2.61 | ×2.18 | ×1.87 | ×1.56 | ×4.67 | ×3.43 | ×2.82 | ×2.44 | ×2.13 | ×1.78 | ×4.45 | ×3.42 | ×2.74 | ×2.28 | ×2.08 | ×1.81 |
| | SqLLM | ×3.69 | ×3.07 | - | - | - | - | ×4.08 | ×3.12 | - | - | - | - | ×4.22 | ×3.17 | - | - | - | - |
| **RTX 4070 Laptop** | Ours | ×4.97 | ×3.73 | ×3.01 | ×2.51 | ×2.10 | ×1.76 | ×5.15 | ×3.84 | ×3.07 | ×2.46 | ×2.06 | ×1.72 | ×5.29 | ×3.66 | ×3.05 | ×2.52 | ×2.13 | ×1.87 |
| | SqLLM | ×4.74 | ×3.66 | - | - | - | - | ×5.05 | ×3.76 | - | - | - | - | ×5.29 | ×3.93 | - | - | - | - |
| **Jetson AGX Orin** | Ours | ×3.84 | ×3.02 | ×2.56 | ×2.33 | ×2.10 | ×1.78 | ×4.20 | ×3.31 | ×2.72 | ×2.47 | ×2.18 | ×1.84 | ×4.35 | ×2.96 | ×2.54 | ×2.52 | ×2.16 | ×1.86 |
| | SqLLM | ×3.15 | ×1.94 | - | - | - | - | ×3.30 | ×1.98 | - | - | - | - | ×3.36 | ×2.04 | - | - | - | - |

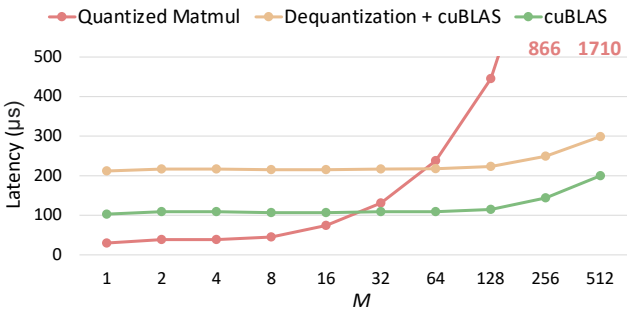


*Figure 11.* Latency comparison of 4-bit quantized matrix-matrix multiplication using our kernel, separate dequantization followed by cuBLAS FP16 kernel, and cuBLAS FP16 kernel on RTX 4090. Matrix dimensions are ($M$, 4096) and (11008, 4096), with $M$ varying from 1 to 512.

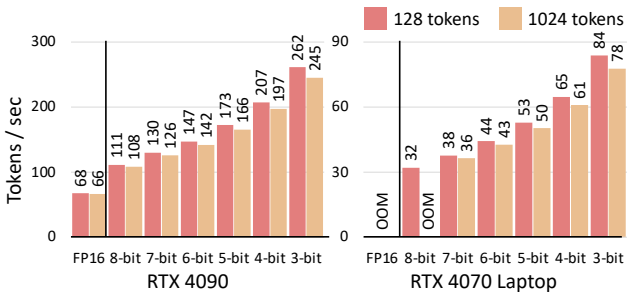


*Figure 12.* End-to-end throughput of Llama-2-7B.

**Matrix-Matrix Multiplication Performance.** Even for our target setting of running a model on personal devices, where a single query is typically served at a time, efficient matrix-matrix multiplication is necessary for two reasons. First, the batch size might not always be 1; it can be small (e.g., 2, 4, 8). Even in a single query, multiple tokens are generated in parallel when using advanced decoding algorithms like beam search (Sutskever et al., 2014) and parallel sampling. Speculative decoding also requires the parallel processing of multiple tokens. Second, during the prefill phase, the tokens in the input prompt are processed together, with their number potentially reaching into the hundreds or thousands.

The red line in Figure 11 illustrates the latency of our kernel for 4-bit quantized matrix-matrix multiplication with dimensions ($M$, 4096) and (11008, 4096) on an RTX 4090, with $M$ varying from 1 to 512. Our kernel performs robustly with small $M$ values such as 2, 4, and 8, showing a significant latency gap compared to the cuBLAS baseline (green line). This indicates that our kernel is also effective for generation with small batch sizes. Similar results are observed for other bit-widths and platforms, as detailed in Appendix C.3.

When $M$ becomes very large, however, as in the prefill phase, our kernel becomes slower than the cuBLAS FP16 baseline (green line). This slowdown occurs because our kernel does not use tensor cores, making it more susceptible to increases in computation with large $M$. To mitigate potential slowdown of the prefill phase, our engine dequantizes the weights using a separate kernel and then employs the cuBLAS kernel when $M$ exceeds a certain threshold (e.g., 16). The latency gap between the dequantization + cuBLAS (represented by the yellow line) and cuBLAS remains almost constant (about 100 µs), as dequantization time does not scale with $M$. This limited performance degradation of the prefill phase is small enough to be easily amortized when the generation length is sufficiently long.

### 6.3. End-to-end Throughput

We evaluate the end-to-end inference throughput of our engine by integrating it with TensorRT-LLM (NVIDIA). Figure 12 shows the throughput of Llama-2-7B in generating 128 and 1024 tokens on RTX 4090 and RTX 4070 Laptop. TensorRT-LLM currently lacks support for Jetson. The enhanced kernel performance effectively translates into an end-to-end speedup. A slightly lower speedup with a longer sequence length (1024) is due to the increased overhead of attention, which is not accelerated by weight quantization. Results for the other models are in Appendix D.

## 7. Conclusion

We make a case for any-precision LLM, which enables memory-efficient and cost-effective deployment of multiple, different-sized LLMs. We propose a lightweight method for any-precision quantization of LLM, along with a specialized software engine to fully leverage its benefits.

## Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

## Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MSIT) (RS-2024-00340008), Institute of Information & Communications Technology Planning & Evaluation (IITP) funded by the Korea Government (MSIT) (2021-0-00105, Development of Model Compression Framework for Scalable On-device AI Computing on Edge Applications), and Institute of Information & Communications Technology Planning & Evaluation (IITP) under the artificial intelligence semiconductor support program (IITP-2023-RS-2023-00256081), funded by the Korea Government (MSIT).

## References

Agarwal, R., Vieillard, N., Zhou, Y., Stanczyk, P., Ramos, S., Geist, M., and Bachem, O. Generalized knowledge distillation for auto-regressive language models. In *The Twelfth International Conference on Learning Representations*, 2024.

Chee, J., Cai, Y., Kuleshov, V., and Sa, C. D. QuIP: 2-bit quantization of large language models with guarantees. *Advances in Neural Information Processing Systems*, 36, 2023.

Chen, C., Borgeaud, S., Irving, G., Lespiau, J.-B., Sifre, L., and Jumper, J. Accelerating large language model decoding with speculative sampling, 2023.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the AI2 reasoning challenge, 2018.

Cowan, M., Moreau, T., Chen, T., Bornholt, J., and Ceze, L. Automatic generation of high-performance quantized machine learning kernels. In *Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization*, CGO 2020, pp. 305–316, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370479. doi: 10.1145/3368826.3377912.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. QLoRA: Efficient finetuning of quantized LLMs. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Dettmers, T., Svirschevski, R., Egiazarian, V., Kuznedelev, D., Frantar, E., Ashkboos, S., Borzunov, A., Hoefler, T., and Alistarh, D. SpQR: A sparse-quantized representation for near-lossless llm weight compression. 2024.

Frantar, E. and Alistarh, D. SparseGPT: Massive language models can be accurately pruned in one-shot. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10323–10337. PMLR, 2023.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. OPTQ: Accurate quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*, 2023.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, 12 2023.

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., and Keutzer, K. A survey of quantization methods for efficient neural network inference, 2021.

Gu, Y., Dong, L., Wei, F., and Huang, M. MiniLLM: Knowledge distillation of large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

Han, S., Shen, H., Philipose, M., Agarwal, S., Wolman, A., and Krishnamurthy, A. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, pp. 123–136, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342698. doi: 10.1145/2906388.2906396.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network, 2015.

Hsieh, C., Li, C., Yeh, C., Nakhost, H., Fujii, Y., Ratner, A., Krishna, R., Lee, C., and Pfister, T. Distilling Step-by-Step! outperforming larger language models with less training data and smaller model sizes. In Rogers, A., Boyd-Graber, J. L., and Okazaki, N. (eds.), *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pp. 8003–8017. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-ACL.507.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. Mistral 7B, 2023.

Kim, J., Lee, J. H., Kim, S., Park, J., Yoo, K. M., Kwon, S. J., and Lee, D. Memory-efficient fine-tuning of compressed large language models via sub-4-bit integer quantization. *Advances in Neural Information Processing Systems*, 36, 2023a.

Kim, S., Hooper, C., Gholami, A., Dong, Z., Li, X., Shen, S., Mahoney, M. W., and Keutzer, K. SqueezeLLM: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*, 2023b.

Kim, S., Hooper, C., Wattanawong, T., Kang, M., Yan, R., Genc, H., Dinh, G., Huang, Q., Keutzer, K., Mahoney, M. W., Shao, Y. S., and Gholami, A. Full stack optimization of transformer inference: a survey, 2023c.

Kim, S., Mangalam, K., Moon, S., Malik, J., Mahoney, M. W., Gholami, A., and Keutzer, K. Speculative decoding with big little decoder. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023d.

Lee, C., Jin, J., Kim, T., Kim, H., and Park, E. Owq: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(12):13355–13364, Mar. 2024. doi: 10.1609/aaai.v38i12.29237.

Leviathan, Y., Kalman, M., and Matias, Y. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., and Han, S. AWQ: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

Liu, Z., Oguz, B., Zhao, C., Chang, E., Stock, P., Mehdad, Y., Shi, Y., Krishnamoorthi, R., and Chandra, V. LLM-QAT: Data-free quantization aware training for large language models, 2023.

Ma, X., Fang, G., and Wang, X. LLM-Pruner: On the structural pruning of large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. The Penn Treebank: annotating predicate argument structure. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, pp. 114–119, USA, 1994. Association for Computational Linguistics. ISBN 1558603573. doi: 10.3115/1075812.1075835.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models, 2016.

Miao, X., Oliaro, G., Zhang, Z., Cheng, X., Wang, Z., Wong, R. Y. Y., Zhu, A., Yang, L., Shi, X., Shi, C., Chen, Z., Arfeen, D., Abhyankar, R., and Jia, Z. SpecInfer: Accelerating generative large language model serving with speculative inference and token tree verification, 2023.

NVIDIA. TensorRT-LLM. URL https://github.com/NVIDIA/TensorRT-LLM.

Park, G., Park, B., Kim, M., Lee, S., Kim, J., Kwon, B., Kwon, S. J., Kim, B., Lee, Y., and Lee, D. LUT-GEMM: Quantized matrix multiplication based on LUTs for efficient inference in large-scale generative language models. In *The Twelfth International Conference on Learning Representations*, 2024.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. WinoGrande: an adversarial winograd schema challenge at scale. *Commun. ACM*, 64(9):99–106, aug 2021. ISSN 0001-0782. doi: 10.1145/3474381.

Santacroce, M., Wen, Z., Shen, Y., and Li, Y. What matters in the structured pruning of generative language models?, 2023.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pp. 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

Tata, S. and Patel, J. M. PiQA: An algebra for querying protein data sets. In *Proceedings of the 15th International Conference on Scientific and Statistical Database Management (SSDBM 2003), 9-11 July 2003, Cambridge, MA, USA*, pp. 141–150. IEEE Computer Society, 2003. doi: 10.1109/SSDM.2003.1214975.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y.,

Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., and Scialom, T. Llama 2: Open foundation and fine-tuned chat models, 2023.

turboderp. ExLlamaV2. URL https://github.com/turboderp/exllamav2.

Umuroglu, Y. and Jahre, M. Work-in-progress: towards efficient quantized neural network inference on mobile devices. In *2017 International Conference on Compilers, Architectures and Synthesis For Embedded Systems (CASES)*, pp. 1–2, 2017. doi: 10.1145/3125501.3125528.

Wan, C., Santriaji, M., Rogers, E., Hoffmann, H., Maire, M., and Lu, S. ALERT: Accurate learning for energy and timeliness. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pp. 353–369. USENIX Association, July 2020. ISBN 978-1-939133-14-4.

Warren, H. S. *Hacker's Delight*. Addison-Wesley Professional, 2nd edition, 2012. ISBN 0321842685.

Yu, H., Li, H., Shi, H., Huang, T. S., and Hua, G. Any-precision deep neural networks. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pp. 10763–10771. AAAI Press, 2021. doi: 10.1609/AAAI.V35I12.17286.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. HellaSwag: Can a machine really finish your sentence? In Korhonen, A., Traum, D., and Màrquez, L. (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472.

Zhang, J., Elnikety, S., Zarar, S., Gupta, A., and Garg, S. Model-Switching: Dealing with fluctuating workloads in Machine-Learning-as-a-Service systems. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*. USENIX Association, July 2020.

Zhang, M., Chen, H., Shen, C., Yang, Z., Ou, L., Yu, X., and Zhuang, B. LoRAPrune: Pruning meets low-rank parameter-efficient fine-tuning, 2023.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., Mihaylov, T., Ott, M., Shleifer, S., Shuster, K., Simig, D., Koura, P. S., Sridhar, A., Wang, T., and Zettlemoyer, L. OPT: Open pre-trained transformer language models, 2022.

Zhao, B., Cui, Q., Song, R., Qiu, Y., and Liang, J. Decoupled knowledge distillation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pp. 11943–11952. IEEE, 2022. doi: 10.1109/CVPR52688.2022.01165.

# A. Evaluation Details

## A.1. Datasets

For WikiText-2, we concatenate on the test set to form a continuous string for perplexity evaluation. For C4, we concatenate samples from the validation set, as using the whole unsampled dataset is infeasible and impractical due to the large size of the dataset. For the Penn Treebank dataset, we concatenate on the test set. In its original state, rare words are replaced with "<unk>" — we modify this by splitting each "<unk>" into five separate characters: "<", "u", "n", "k", and ">". This approach prevents the tokenizer from recognizing "<unk>" as a special unknown token, which would significantly increase model perplexity due to the inability to predict occurrences of unknown words. By treating each character of "<unk>" as a regular token, we maintain consistent token processing.

## A.2. Perplexity Calculations

To evaluate a given string, we start by tokenizing it with the default HuggingFace tokenizer of each model. Then we chunk the sequence into non-overlapping segments of length 2048, as in the manner of previous works (Frantar et al., 2023) (Kim et al., 2023b). We process these segments through the model to collect the log-probabilities of subsequent token generation. The final perplexity we report is the exponentiated average of these log-probabilities.

## A.3. Zero-shot Tasks

We use the Language Model Evaluation Harness framework (Gao et al., 2023) to evaluate our models on zero-shot tasks. We report the byte-length normalized accuracy where applicable.

# B. Full Results on Zero-shot Tasks

The data in Table 8 presents the full results for each of the five zero-shot tasks (ARC-easy, ARC-challenge, HellaSwag, PIQA, WinoGrande). The results indicate that the SqLLM models with incremental upscaling (SqLLM+IU) match the independently quantized SqLLM models across a range of tasks, models, and bit-widths.

# C. Additional Kernel Microbenchmark Results

## C.1. Kernel Latency on Various Matrix Sizes

Table 9 presents the latency of of our kernel for matrix-vector multiplication across various matrix sizes. We select the matrix sizes that are used in Llama-2-7B, Mistral-7B, OPT-6.7B, OPT-2.7B and OPT-1.3B.

## C.2. Comparison with Kernels for Uniform Quantization

Table 10 provides a comparison of our kernel's matrix-vector multiplication performance against existing kernels designed for uniform quantization: ExLlamaV2 (turboderp), LUT-GEMM (Park et al., 2024), AWQ (Lin et al., 2023), TensorRT-LLM (NVIDIA). Note that this is not an apples-to-apples comparison as they lack any-precision support. They are specifically tailored for uniform quantization and/or employ a bitpacking-based weight representation. Nevertheless, our kernel demonstrates competitive performance, achieving the best results in the majority of cases.

## C.3. Matrix-Matrix Multiplication Performance

Table 11 shows the matrix-matrix multiplication performance with small batch sizes (2, 4, 8) on our kernel to test its applicability on the small-batch use cases mentioned above. Our kernel demonstrates impressive performance for multiple batches on both the RTX 4090 and RTX 4070 Laptop, with less than a 30% drop in performance compared to single-batch processing in most scenarios. The exceptions are in cases with very low bit-widths, like 3 and 4-bit, where the performance difference might be more pronounced. This trend slightly varies for Jetson AGX Orin, which has highly limited compute resources, but still delivers commendable results, particularly for batch sizes of 2 and 4.

# D. Additional End-to-End Throughput Evaluation Results

Figure 13 shows the end-to-end inference throughput of Mistral-7B, OPT-6.7B, OPT-2.7B and OPT-1.3B.

# E. Incremental Upscaling with Uniform Quantization Methods

In this section, we elaborate on our endeavor to apply incremental upscaling to the two state-of-the-art uniform quantization methods (GPTQ, AWQ), which ultimately results in failure. Figure 14 illustrates the process of incremental upscaling with uniform quantization methods. Upscaling on uniform quantization methods requires that each bin in the quantization grid be divided equally; this is in contrast with non-uniform quantization, where each bin in the quantization grid can be divided into two new bins of arbitrary sizes. Naturally, each bin's midpoint becomes its representative value. With these premises, Appendix E.1 and E.2 discuss issues with upscaling GPTQ and AWQ, respectively.

## E.1. Incremental Upscaling with GPTQ

Algorithm 1 presents a modified version of GPTQ that additionally includes a clamping operation to preserve the essen-

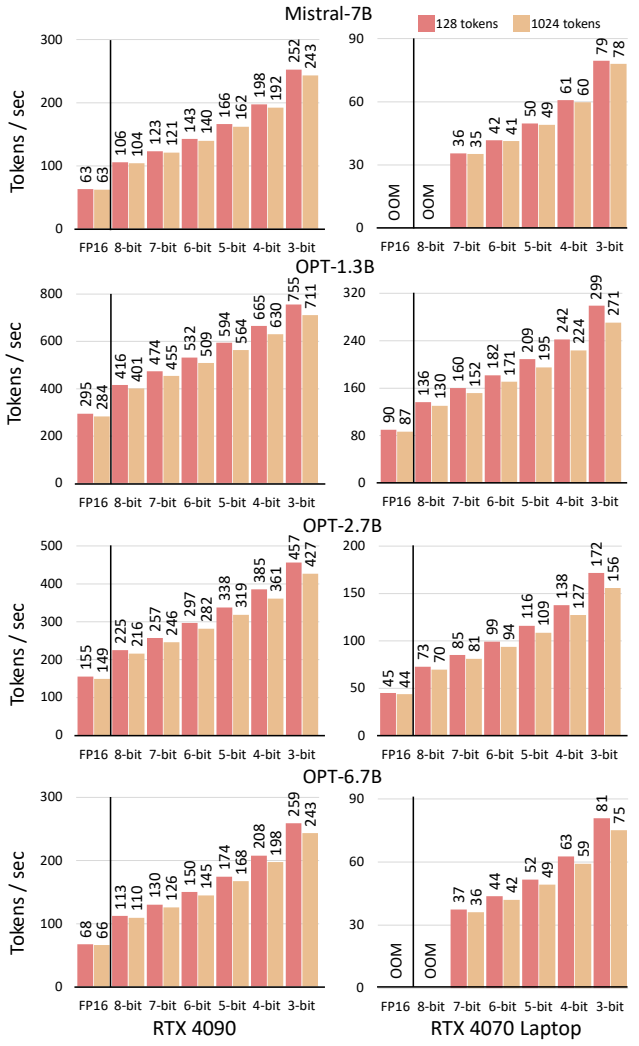


*Figure 13.* End-to-end throughput of Mistral-7B, OPT-6.7B, OPT-2.7B and OPT-1.3B.

tial weight-inheriting characteristic of the upscaling process. While clamping is necessary, it detrimentally forces weights to suboptimal values — hence it is desirable for clamping to occur sparingly. As the algorithm processes each column of the weight matrix, it updates the error-compensated weights $W_{n+1}$ and, in correlation, generates the quantized weights $Q_{n+1}$. Therefore, to limit clamping, $W_{n+1}$ must evolve similarly to $W_n$, the error-compensated weights of the previous $n$-bit quantization, ensuring that $Q_{n+1}$ remains closely related to $Q_n$. However, this is not the observed case. $W_n$ and $W_{n+1}$, despite both starting as the original weight $W$, soon begin to diverge. The rounding functions $RTN_n$ and $RTN_{n+1}$ produce subtly different outcomes, leading to varied compensation for $W_n$ and $W_{n+1}$. Initially minor, this divergence grows over time, eventually necessitating clamping. Clamping then triggers a positive feedback

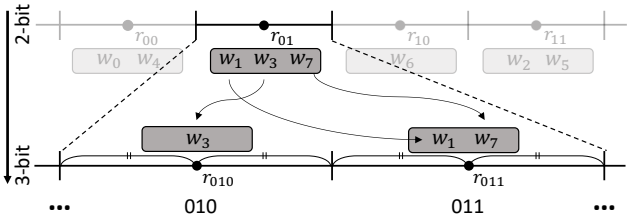


*Figure 14.* Incremental upscaling from 2-bit to 3-bit with uniform quantization methods.

**Algorithm 1** Incremental Upscaling of GPTQ

**Input:** $\mathbf{W}$, $\mathbf{Q}_n$ // original weight, $n$-bit quantized weight
**Output:** $\mathbf{Q}_{n+1}$ // ($n$+1)-bit quantized weight

$\mathbf{Q}_{n+1} = \mathbf{0}_{K\times N}$
$\mathbf{W}_{n+1} = \mathbf{W}$
**for** $i = 0, ..., N$ **do**
  $\mathbf{Q}_{n+1}[:, i] \leftarrow RTN_{n+1}(\mathbf{W}_{n+1}[:, i])$
  $\mathbf{Q}_{n+1}[:, i] \leftarrow clamp(\mathbf{Q}_{n+1}[:, i], \mathbf{Q}_n[:, i] \times 2, \mathbf{Q}_n[:, i] \times 2 + 1)$
  $\mathbf{E} \leftarrow compute\_err(\mathbf{Q}_{n+1}[:, i], \mathbf{W}_{n+1}[:, i])$
  $\mathbf{W}_{n+1}[:, i:] \leftarrow compensate\_err(\mathbf{W}_{n+1}[:, i:], \mathbf{E})$
**return** $\mathbf{Q}_{n+1}$

loop, exacerbating disturbances in the upscaling process and further increasing clamping occurrences.

In Figure 15 we have randomly sampled six different weight matrices to demonstrate this runaway clamping effect. The error-compensated weight values $W_3$ and $W_4$ evolve differently in the seed 3-bit and upscaled 4-bit quantizations of the Llama-2-7B model. As the algorithm iterates over the columns, we plot the root-mean-square deviation (RMSD) of $W_4$ against $W_3$ on the left axis, along with the average clamping amount per weight value on the right axis. For all six matrices, the divergence between $W_3$ and $W_4$ starts small, but once the clamping amount starts to grow, it does so exponentially — eventually leading to drastic errors.

The resulting instability of the upscaling process leads to the large quality drops shown in Table 7, making GPTQ unfavorable for any-precison support. GPTQ-R refers to GPTQ with activation reordering.

### E.2. Incremental Upscaling with AWQ

The AWQ process comprises two main steps, as depicted in Figure 16(a). Initially, weight parameters, denoted as $\mathbb{W}$, undergo a preprocessing phase to become $\mathbb{W}_{\mathbb{m}}$, involving channel-wise scaling and clipping adjustments tailored to the rounding function in use. For instance, in 3-bit quantization, the preprocessing involves $\mathbb{W}$ and a specific 3-bit rounding function, $RTN_3$, which segments the value range

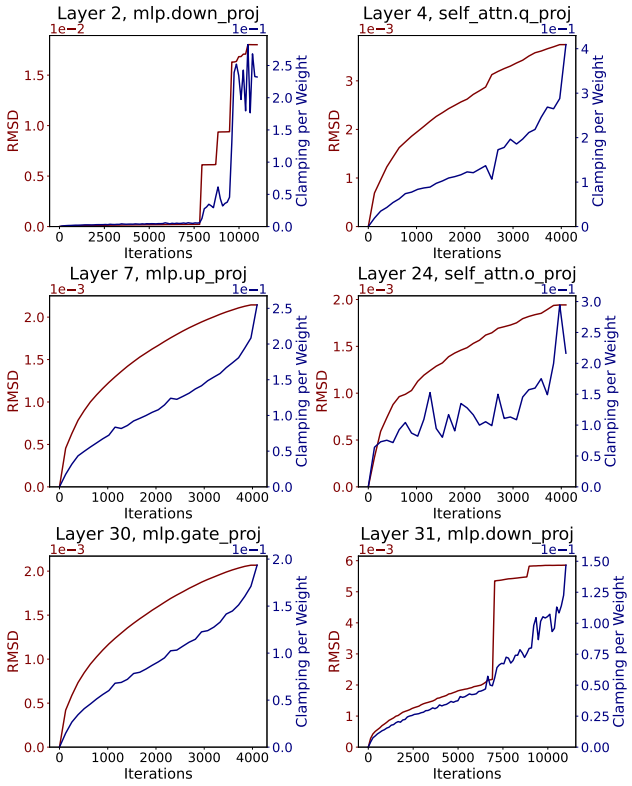


*Figure 15.* RMSD between the error-compensated weight matrices $W_3$ and $W_4$, along with the average clamping amount per weight, as the modified GPTQ algorithm progresses along the columns of Llama-2-7B.

into $2^3$ equal parts for rounding. This step includes a grid search to determine the optimal scaling and clipping for minimal error post-rounding. Once preprocessing is finalized, applying $RTN_3$ to $\mathbb{W}_\mathbb{n}$ yields the quantized weights, $\mathbb{Q}_3$. This methodology is adaptable for other bit-widths by substituting the appropriate rounding function, like $RTN_4$ or $RTN_5$, to suit the desired quantization precision.

When applying upscaling to AWQ, however, the preprocessing step used for the initial 3-bit model is reused across all bit-widths during upscaling, rather than tailoring preprocessing to each specific bit-width. This approach is depicted in Figure 16(b) and is necessary to support any-precision quantization. For instance, upscaling from a 3-bit to a 4-bit model involves using the preprocessing result $\mathbb{W}_3$ from the 3-bit model and applying a function, $RTN_{3\to4}$, that redistributes values into $2^4$ bins by further dividing each 3-bit bin. This uniform preprocessing strategy is maintained for all upscaling steps to higher bit-widths. Although this method ensures consistency across quantizations, it relies on preprocessing factors optimized only for the seed model, which may not be ideal for upscaled models. The rounding functions used during upscaling, such as $RTN_{3\to4}$ and

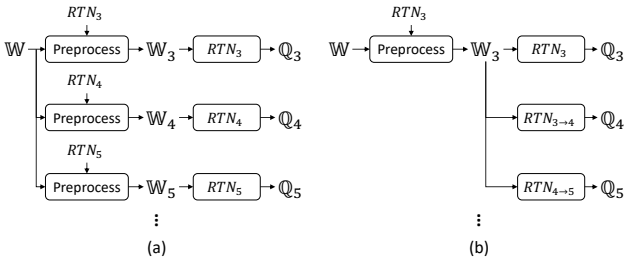


*Figure 16.* High-level abstraction of (a) original AWQ method and (b) AWQ with incremental upscaling.

$RTN_{4\to5}$, differ from the seed model's $RTN_3$ in both the number of bins, the distribution of values, as well as whether a zero-point exists, which can lead to performance degradation as demonstrated in Table 7.

*Table 7.* Perplexity measured on Wikitext2, C4, and Penn Treebank for vanilla GPTQ, GPTQ-R and AWQ, as well as for their counterparts integrated with incremental scaling—namely, GPTQ+IU, GPTQ-R+IU, and AWQ+IU, respectively. 3-bit models are used as the seed models.

| | | 3-bit | | | 4-bit | | | 5-bit | | | 6-bit | | | 7-bit | | | 8-bit | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Wiki | PTB | C4 | Wiki | PTB | C4 | Wiki | PTB | C4 | Wiki | PTB | C4 | Wiki | PTB | C4 | Wiki | PTB | C4 |
| **Llama-2-7B** | GPTQ | 6.67 | 9.74 | 8.88 | 5.71 | 8.56 | 7.55 | 5.52 | 8.18 | 7.32 | 5.49 | 8.11 | 7.28 | 5.47 | 8.10 | 7.27 | 5.47 | 8.10 | 7.26 |
| | GPTQ+IU | - | - | - | 36.82 | 30.04 | 63.19 | 6.03 | 9.27 | 8.36 | 5.93 | 9.33 | 8.11 | 5.81 | 8.96 | 7.83 | 5.74 | 8.73 | 7.66 |
| | GPTQ-R | 6.73 | 11.36 | 8.92 | 5.68 | 8.39 | 7.52 | 5.52 | 8.16 | 7.32 | 5.48 | 8.11 | 7.28 | 5.47 | 8.10 | 7.27 | 5.47 | 8.10 | 7.26 |
| | GPTQ-R+IU | - | - | - | 12.70 | 22.54 | 20.81 | 6.00 | 9.14 | 7.99 | 5.79 | 8.68 | 7.67 | 5.74 | 8.61 | 7.60 | 5.73 | 8.59 | 7.59 |
| | AWQ | 6.24 | 9.03 | 8.30 | 5.59 | 8.27 | 7.44 | 5.50 | 8.14 | 7.31 | 5.48 | 8.11 | 7.28 | 5.47 | 8.09 | 7.27 | 5.47 | 8.10 | 7.26 |
| | AWQ+IU | - | - | - | INF | INF | INF | 22.50 | 39.37 | INF | 8.16 | 12.22 | 11.42 | 6.07 | 9.01 | 8.07 | 5.67 | 8.34 | 7.51 |
| **Mistral-7B** | GPTQ | 8.93 | 12.02 | 13.19 | 5.43 | 8.78 | 8.64 | 5.29 | 8.52 | 8.45 | 5.26 | 8.49 | 8.40 | 5.25 | 8.46 | 8.38 | 5.25 | 8.46 | 8.38 |
| | GPTQ+IU | - | - | - | 16.73 | 32.65 | 20.67 | 5.93 | 10.58 | 9.10 | 5.47 | 8.86 | 8.61 | 5.42 | 8.71 | 8.54 | 5.40 | 8.68 | 8.52 |
| | GPTQ-R | 6.47 | 10.85 | 9.74 | 5.40 | 8.67 | 8.60 | 5.27 | 8.52 | 8.43 | 5.25 | 8.47 | 8.39 | 5.25 | 8.46 | 8.38 | 5.25 | 8.46 | 8.38 |
| | GPTQ-R+IU | - | - | - | 15.69 | 30.34 | 26.39 | 5.71 | 9.17 | 8.89 | 5.40 | 8.77 | 8.61 | 5.36 | 8.69 | 8.56 | 5.35 | 8.67 | 8.54 |
| **OPT-6.7B** | AWQ | 5.90 | 9.55 | 9.28 | 5.35 | 8.68 | 8.55 | 5.27 | 8.51 | 8.42 | 5.25 | 8.47 | 8.39 | 5.25 | 8.46 | 8.38 | 5.25 | 8.46 | 8.38 |
| | AWQ+IU | - | - | - | 20.07 | 45.75 | 32.88 | 7.68 | 15.33 | 12.78 | 5.83 | 9.68 | 9.24 | 5.42 | 8.79 | 8.62 | 5.34 | 8.62 | 8.52 |
| | GPTQ | 11.58 | 13.43 | 13.94 | 10.90 | 12.64 | 13.25 | 10.80 | 12.59 | 13.09 | 10.77 | 12.55 | 13.07 | 10.81 | 12.52 | 13.05 | 10.84 | 12.52 | 13.05 |
| | GPTQ+IU | - | - | - | 12.40 | 14.13 | 14.44 | 11.18 | 12.77 | 13.33 | 10.95 | 12.62 | 13.15 | 10.86 | 12.62 | 13.12 | 10.84 | 12.59 | 13.09 |
| | GPTQ-R | 11.70 | 13.75 | 13.94 | 11.03 | 12.79 | 13.20 | 10.86 | 12.59 | 13.07 | 10.84 | 12.57 | 13.05 | 10.84 | 12.55 | 13.05 | 10.84 | 12.52 | 13.05 |
| | GPTQ-R+IU | - | - | - | 12.52 | 13.99 | 14.38 | 11.23 | 12.95 | 13.35 | 11.03 | 12.87 | 13.23 | 10.96 | 12.84 | 13.20 | 10.96 | 12.84 | 13.20 |
| | AWQ | 11.38 | 13.30 | 13.73 | 10.96 | 12.72 | 13.17 | 10.86 | 12.57 | 13.07 | 10.86 | 12.55 | 13.07 | 10.86 | 12.55 | 13.05 | 10.86 | 12.55 | 13.05 |
| | AWQ+IU | - | - | - | INF | INF | INF | INF | INF | INF | INF | INF | INF | INF | 55360.00 | INF | 27632.00 | 17296.00 | 28512.00 |
| **OPT-2.7B** | GPTQ | 14.24 | 16.88 | 16.55 | 12.62 | 15.07 | 15.10 | 12.30 | 14.58 | 14.81 | 12.26 | 14.49 | 14.75 | 12.26 | 14.44 | 14.75 | 12.23 | 14.44 | 14.73 |
| | GPTQ+IU | - | - | - | 14.08 | 17.59 | 16.98 | 12.70 | 15.05 | 15.16 | 12.42 | 14.73 | 14.90 | 12.33 | 14.58 | 14.84 | 12.33 | 14.55 | 14.81 |
| | GPTQ-R | 14.52 | 17.34 | 16.36 | 12.47 | 14.87 | 14.98 | 12.26 | 14.49 | 14.81 | 12.20 | 14.44 | 14.75 | 12.23 | 14.44 | 14.75 | 12.23 | 14.44 | 14.75 |
| | GPTQ-R+IU | - | - | - | 14.35 | 16.33 | 16.42 | 12.55 | 14.90 | 15.10 | 12.35 | 14.64 | 14.90 | 12.35 | 14.58 | 14.87 | 12.38 | 14.55 | 14.84 |
| | AWQ | 13.56 | 16.14 | 15.95 | 12.70 | 14.66 | 14.98 | 12.52 | 14.47 | 14.81 | 12.49 | 14.44 | 14.75 | 12.49 | 14.44 | 14.75 | 12.47 | 14.44 | 14.75 |
| | AWQ+IU | - | - | - | INF | INF | INF | INF | 54080.00 | INF | 48864.00 | 50432.00 | 41152.00 | 37184.00 | 38976.00 | 31792.00 | 18400.00 | 16112.00 | 18256.00 |
| **OPT-1.3B** | GPTQ | 17.48 | 19.66 | 19.17 | 15.05 | 16.88 | 17.19 | 14.93 | 16.59 | 17.25 | 14.70 | 16.30 | 16.81 | 14.75 | 16.33 | 16.91 | 14.81 | 16.36 | 16.95 |
| | GPTQ+IU | - | - | - | 16.98 | 18.88 | 19.05 | 15.37 | 16.98 | 17.25 | 14.98 | 16.66 | 16.91 | 14.90 | 16.55 | 16.81 | 14.90 | 16.55 | 16.78 |
| | GPTQ-R | 16.98 | 18.91 | 18.61 | 14.95 | 16.63 | 16.91 | 14.70 | 16.36 | 16.66 | 14.70 | 16.27 | 16.59 | 14.66 | 16.23 | 16.59 | 14.64 | 16.23 | 16.59 |
| | GPTQ-R+IU | - | - | - | 16.39 | 18.00 | 18.28 | 15.22 | 16.75 | 17.08 | 15.02 | 16.66 | 16.91 | 14.93 | 16.52 | 16.88 | 14.90 | 16.52 | 16.88 |
| | AWQ | 16.33 | 18.58 | 18.33 | 14.95 | 16.69 | 16.91 | 14.64 | 16.30 | 16.63 | 14.64 | 16.23 | 16.59 | 14.64 | 16.20 | 16.55 | 14.61 | 16.20 | 16.55 |
| | AWQ+IU | - | - | - | 7672.00 | 4074.00 | INF | 5192.00 | 3174.00 | INF | 3540.00 | 1970.00 | 2652.00 | 1559.00 | 780.50 | 1243.00 | 424.50 | 149.63 | 361.75 |

*Table 8.* Zero-shot accuracies on five tasks — ARC-easy (ARC-e), ARC-challenge (ARC-c), HellaSwag, PIQA, and WinoGrande — for vanilla SqueezeLLM (SqLLM) and SqueezeLLM integrated with incremental upscaling (SqLLM+IU) using a 3-bit seed model. We also report the difference between the two methods, and calculate the average accuracy across the five tasks.

| | | 3-bit | | | | | | 4-bit | | | | | | 5-bit | | | | | | FP16 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ARC-e | ARC-c | HellaSwag | PIQA | WinoGrande | Average | ARC-e | ARC-c | HellaSwag | PIQA | WinoGrande | Average | ARC-e | ARC-c | HellaSwag | PIQA | WinoGrande | Average | ARC-e | ARC-c | HellaSwag | PIQA | WinoGrande | Average |
| **Llama-2-7B** | SqLLM | 70.3 % | 41.8 % | 73.0 % | 77.9 % | 67.8 % | 66.2 % | 73.8 % | 45.5 % | 75.3 % | 78.8 % | 68.0 % | 68.3 % | 74.3 % | 45.4 % | 75.8 % | 78.8 % | 68.8 % | 68.6 % | | | | | | |
| | SqLLM+IU | - | - | - | - | - | - | 73.5 % | 44.2 % | 75.3 % | 78.7 % | 69.1 % | 68.2 % | 74.5 % | 45.8 % | 75.8 % | 78.9 % | 68.9 % | 68.8 % | | | | | | |
| | Δ | - | - | - | - | - | - | -0.3 pp | -1.3 pp | +0.0 pp | -0.0 pp | +1.1 pp | -0.1 pp | +0.1 pp | +0.4 pp | +0.1 pp | +0.1 pp | +0.1 pp | +0.2 pp | | | | | | |
| | | 6-bit | | | | | | 7-bit | | | | | | 8-bit | | | | | | 74.6 % | 46.2 % | 76.0 % | 79.1 % | 69.0 % | 69.0 % |
| | SqLLM | 74.5 % | 46.2 % | 75.9 % | 78.8 % | 68.8 % | 68.8 % | 74.6 % | 45.9 % | 76.0 % | 79.2 % | 69.1 % | 68.9 % | 74.4 % | 46.1 % | 76.0 % | 79.1 % | 69.0 % | 68.9 % | | | | | | |
| | SqLLM+IU | 74.6 % | 46.2 % | 76.1 % | 79.1 % | 68.6 % | 68.9 % | 74.4 % | 46.1 % | 76.0 % | 79.1 % | 69.1 % | 68.9 % | 74.3 % | 46.3 % | 76.0 % | 79.1 % | 69.1 % | 69.0 % | | | | | | |
| | Δ | +0.1 pp | 0.0 pp | +0.2 pp | +0.2 pp | -0.2 pp | +0.1 pp | -0.2 pp | +0.2 pp | -0.0 pp | -0.0 pp | +0.1 pp | 0.0 pp | -0.0 pp | +0.3 pp | -0.0 pp | 0.0 pp | +0.1 pp | +0.1 pp | | | | | | |
| | | 3-bit | | | | | | 4-bit | | | | | | 5-bit | | | | | | FP16 | | | | | |
| | SqLLM | 76.0 % | 49.4 % | 78.1 % | 81.6 % | 72.5 % | 71.5 % | 77.9 % | 52.1 % | 80.3 % | 81.6 % | 73.9 % | 73.2 % | 79.1 % | 53.7 % | 80.9 % | 81.8 % | 73.7 % | 73.8 % | | | | | | |
| | SqLLM+IU | - | - | - | - | - | - | 78.2 % | 52.3 % | 80.3 % | 81.9 % | 74.0 % | 73.3 % | 79.2 % | 53.5 % | 80.8 % | 82.2 % | 73.6 % | 73.8 % | | | | | | |
| **Mistral-7B** | Δ | - | - | - | - | - | - | +0.3 pp | +0.2 pp | -0.0 pp | +0.4 pp | +0.1 pp | +0.1 pp | +0.1 pp | -0.2 pp | -0.1 pp | +0.3 pp | -0.2 pp | 0.0 pp | | | | | | |
| | | 6-bit | | | | | | 7-bit | | | | | | 8-bit | | | | | | 79.6 % | 53.9 % | 81.0 % | 82.1 % | 73.9 % | 74.1 % |
| | SqLLM | 79.5 % | 53.4 % | 81.0 % | 82.1 % | 74.1 % | 74.0 % | 79.5 % | 53.8 % | 81.0 % | 82.2 % | 73.8 % | 74.1 % | 79.6 % | 54.1 % | 81.1 % | 82.2 % | 73.9 % | 74.2 % | | | | | | |
| | SqLLM+IU | 79.4 % | 53.7 % | 81.0 % | 82.0 % | 73.6 % | 74.0 % | 79.4 % | 54.2 % | 81.1 % | 82.3 % | 73.9 % | 74.2 % | 79.3 % | 53.9 % | 81.1 % | 82.1 % | 74.0 % | 74.1 % | | | | | | |
| | Δ | -0.0 pp | +0.3 pp | 0.0 pp | -0.1 pp | -0.5 pp | 0.0 pp | -0.1 pp | +0.3 pp | +0.0 pp | +0.1 pp | +0.1 pp | +0.1 pp | -0.3 pp | -0.2 pp | -0.0 pp | -0.1 pp | +0.1 pp | -0.1 pp | | | | | | |
| | | 3-bit | | | | | | 4-bit | | | | | | 5-bit | | | | | | FP16 | | | | | |
| | SqLLM | 58.8 % | 31.9 % | 63.8 % | 75.4 % | 63.2 % | 58.6 % | 59.9 % | 33.7 % | 66.2 % | 76.1 % | 63.9 % | 59.9 % | 60.4 % | 34.3 % | 66.9 % | 76.4 % | 65.1 % | 60.6 % | | | | | | |
| | SqLLM+IU | - | - | - | - | - | - | 60.1 % | 33.2 % | 66.0 % | 76.2 % | 66.1 % | 60.3 % | 60.2 % | 34.6 % | 67.0 % | 76.5 % | 64.7 % | 60.6 % | | | | | | |
| **OPT-6.7B** | Δ | - | - | - | - | - | - | +0.2 pp | -0.5 pp | -0.2 pp | +0.1 pp | +2.3 pp | +0.4 pp | -0.2 pp | +0.3 pp | +0.1 pp | +0.1 pp | -0.4 pp | 0.0 pp | | | | | | |
| | | 6-bit | | | | | | 7-bit | | | | | | 8-bit | | | | | | 60.1 % | 34.6 % | 67.2 % | 76.7 % | 65.3 % | 60.8 % |
| | SqLLM | 59.9 % | 34.7 % | 67.2 % | 76.7 % | 65.3 % | 60.7 % | 60.1 % | 34.6 % | 67.2 % | 76.4 % | 65.5 % | 60.7 % | 60.0 % | 34.7 % | 67.3 % | 76.6 % | 65.7 % | 60.8 % | | | | | | |
| | SqLLM+IU | 60.3 % | 34.9 % | 67.3 % | 76.4 % | 64.6 % | 60.7 % | 60.1 % | 34.6 % | 67.2 % | 76.7 % | 65.5 % | 60.8 % | 60.1 % | 34.6 % | 67.2 % | 76.6 % | 65.0 % | 60.7 % | | | | | | |
| | Δ | +0.4 pp | +0.2 pp | +0.2 pp | -0.3 pp | -0.6 pp | 0.0 pp | -0.0 pp | 0.0 pp | +0.0 pp | +0.3 pp | 0.0 pp | +0.1 pp | +0.2 pp | -0.1 pp | -0.0 pp | 0.0 pp | -0.7 pp | -0.1 pp | | | | | | |
| | | 3-bit | | | | | | 4-bit | | | | | | 5-bit | | | | | | FP16 | | | | | |
| | SqLLM | 53.2 % | 30.5 % | 56.0 % | 72.6 % | 59.4 % | 54.3 % | 53.5 % | 31.3 % | 59.2 % | 74.2 % | 60.6 % | 55.8 % | 53.9 % | 31.2 % | 60.1 % | 74.8 % | 61.3 % | 56.3 % | | | | | | |
| | SqLLM+IU | - | - | - | - | - | - | 54.4 % | 31.1 % | 59.0 % | 74.1 % | 60.1 % | 55.7 % | 54.3 % | 30.7 % | 60.0 % | 74.4 % | 60.9 % | 56.1 % | | | | | | |
| **OPT-2.7B** | Δ | - | - | - | - | - | - | +0.9 pp | -0.3 pp | -0.3 pp | -0.1 pp | -0.6 pp | -0.1 pp | +0.4 pp | -0.5 pp | -0.0 pp | -0.4 pp | -0.5 pp | -0.2 pp | | | | | | |
| | | 6-bit | | | | | | 7-bit | | | | | | 8-bit | | | | | | 54.4 % | 31.2 % | 60.7 % | 74.8 % | 60.8 % | 56.4 % |
| | SqLLM | 54.1 % | 31.2 % | 60.4 % | 74.7 % | 60.9 % | 56.2 % | 54.4 % | 30.9 % | 60.5 % | 74.7 % | 61.2 % | 56.3 % | 54.2 % | 31.1 % | 60.6 % | 74.8 % | 61.0 % | 56.3 % | | | | | | |
| | SqLLM+IU | 54.3 % | 31.0 % | 60.4 % | 74.8 % | 60.5 % | 56.2 % | 54.3 % | 31.5 % | 60.5 % | 74.6 % | 61.2 % | 56.4 % | 54.3 % | 31.3 % | 60.6 % | 74.8 % | 60.8 % | 56.3 % | | | | | | |
| | Δ | +0.1 pp | -0.3 pp | -0.0 pp | +0.1 pp | -0.3 pp | 0.0 pp | -0.1 pp | +0.6 pp | +0.0 pp | -0.1 pp | 0.0 pp | +0.1 pp | +0.1 pp | +0.3 pp | -0.0 pp | 0.0 pp | -0.2 pp | 0.0 pp | | | | | | |
| | | 3-bit | | | | | | 4-bit | | | | | | 5-bit | | | | | | FP16 | | | | | |
| | SqLLM | 48.3 % | 27.5 % | 49.6 % | 71.8 % | 56.3 % | 50.7 % | 51.5 % | 28.5 % | 52.9 % | 71.2 % | 59.3 % | 52.7 % | 51.2 % | 29.0 % | 53.6 % | 71.9 % | 59.0 % | 52.9 % | | | | | | |
| | SqLLM+IU | - | - | - | - | - | - | 49.8 % | 29.9 % | 53.0 % | 71.7 % | 58.6 % | 52.6 % | 50.7 % | 29.6 % | 53.5 % | 72.4 % | 59.1 % | 53.1 % | | | | | | |
| **OPT-1.3B** | Δ | - | - | - | - | - | - | -1.7 pp | +1.4 pp | +0.0 pp | +0.4 pp | -0.7 pp | -0.1 pp | -0.5 pp | +0.6 pp | -0.0 pp | +0.5 pp | +0.2 pp | +0.2 pp | | | | | | |
| | | 6-bit | | | | | | 7-bit | | | | | | 8-bit | | | | | | 51.0 % | 29.6 % | 53.7 % | 72.5 % | 60.0 % | 53.3 % |
| | SqLLM | 51.4 % | 30.0 % | 53.7 % | 72.4 % | 59.4 % | 53.4 % | 50.9 % | 29.7 % | 53.7 % | 72.4 % | 60.1 % | 53.4 % | 51.0 % | 29.5 % | 53.7 % | 72.4 % | 59.9 % | 53.3 % | | | | | | |
| | SqLLM+IU | 51.0 % | 29.7 % | 53.7 % | 72.5 % | 59.4 % | 53.3 % | 50.9 % | 29.8 % | 53.7 % | 72.6 % | 60.2 % | 53.4 % | 51.1 % | 29.8 % | 53.7 % | 72.4 % | 59.4 % | 53.3 % | | | | | | |
| | Δ | -0.4 pp | -0.3 pp | 0.0 pp | +0.1 pp | 0.0 pp | -0.1 pp | -0.0 pp | +0.1 pp | -0.0 pp | +0.3 pp | +0.1 pp | 0.0 pp | +0.1 pp | +0.3 pp | -0.0 pp | 0.0 pp | -0.5 pp | 0.0 pp | | | | | | |

*Table 9.* Latency of our kernel on matrix-vector multiplication of various dimensions.

| GPU | Model | Matrix Size | 3-bit | 4-bit | 5-bit | 6-bit | 7-bit | 8-bit | FP16 |
|---|---|---|---|---|---|---|---|---|---|
| RTX 4090 | OPT-1.3B | 2048 x 2048 | 4.73 | 5.95 | 6.63 | 7.13 | 7.95 | 9.06 | 12.16 |
| | | 8192 x 2048 | 9.72 | 12.65 | 14.82 | 17.82 | 22.42 | 27.92 | 38.66 |
| | | 2048 x 8192 | 9.78 | 12.88 | 15.60 | 18.15 | 21.45 | 24.14 | 42.62 |
| | OPT-2.7B | 2560 x 2560 | 5.89 | 7.45 | 8.26 | 9.00 | 10.16 | 11.77 | 19.55 |
| | | 10240 x 2560 | 14.29 | 20.34 | 24.30 | 28.26 | 32.81 | 38.80 | 58.50 |
| | | 2560 x 10240 | 13.88 | 18.83 | 22.68 | 26.80 | 37.60 | 40.59 | 60.90 |
| | Llama-2-7B | 4096 x 4096 | 9.67 | 12.73 | 14.80 | 17.72 | 20.64 | 24.69 | 38.59 |
| | | 11008 x 4096 | 21.90 | 29.80 | 36.26 | 41.95 | 48.13 | 57.44 | 102.34 |
| | | 4096 x 11008 | 23.16 | 30.09 | 37.56 | 45.19 | 49.55 | 56.98 | 103.04 |
| | Mistral-7B | 14336 x 4096 | 28.73 | 39.00 | 46.13 | 52.06 | 62.16 | 74.44 | 132.38 |
| | | 4096 x 14336 | 27.69 | 36.43 | 43.65 | 50.57 | 61.83 | 69.77 | 134.78 |
| | OPT-6.7B | 16384 x 4096 | 30.72 | 42.12 | 49.84 | 59.13 | 69.99 | 86.55 | 150.34 |
| | | 4096 x 16384 | 30.80 | 40.63 | 48.83 | 57.47 | 69.96 | 79.35 | 156.70 |
| RTX 4070 Laptop | OPT-1.3B | 2048 x 2048 | 9.64 | 12.63 | 15.04 | 17.79 | 20.85 | 24.79 | 36.74 |
| | | 8192 x 2048 | 28.93 | 38.26 | 47.46 | 58.08 | 71.17 | 87.97 | 142.05 |
| | | 2048 x 8192 | 29.27 | 39.49 | 48.17 | 56.01 | 68.52 | 77.58 | 145.76 |
| | OPT-2.7B | 2560 x 2560 | 13.92 | 18.94 | 22.56 | 26.00 | 29.94 | 35.26 | 57.66 |
| | | 10240 x 2560 | 43.90 | 60.75 | 75.12 | 90.90 | 107.90 | 134.34 | 219.30 |
| | | 2560 x 10240 | 42.78 | 58.30 | 71.78 | 85.25 | 102.77 | 116.97 | 220.10 |
| | Llama-2-7B | 4096 x 4096 | 28.57 | 38.12 | 47.16 | 56.55 | 67.48 | 80.65 | 142.05 |
| | | 11008 x 4096 | 71.71 | 96.22 | 120.09 | 149.99 | 179.50 | 214.07 | 369.12 |
| | | 4096 x 11008 | 72.21 | 104.42 | 125.35 | 151.54 | 179.48 | 204.47 | 382.21 |
| | Mistral-7B | 14336 x 4096 | 92.42 | 124.22 | 161.02 | 193.62 | 232.61 | 276.84 | 478.08 |
| | | 4096 x 14336 | 92.53 | 134.28 | 164.77 | 197.07 | 223.79 | 258.43 | 484.10 |
| | OPT-6.7B | 16384 x 4096 | 105.06 | 145.05 | 182.29 | 220.29 | 261.84 | 311.81 | 545.12 |
| | | 4096 x 16384 | 104.97 | 151.25 | 191.31 | 224.63 | 253.72 | 291.27 | 546.66 |
| Jetson AGX Orin | OPT-1.3B | 2048 x 2048 | 20.81 | 25.88 | 28.88 | 31.58 | 34.70 | 39.86 | 57.66 |
| | | 8192 x 2048 | 58.37 | 72.80 | 85.00 | 94.15 | 106.76 | 126.68 | 198.18 |
| | | 2048 x 8192 | 53.22 | 67.72 | 81.98 | 88.53 | 108.69 | 112.26 | 207.14 |
| | OPT-2.7B | 2560 x 2560 | 30.49 | 38.88 | 43.60 | 47.57 | 51.97 | 58.90 | 84.83 |
| | | 10240 x 2560 | 92.90 | 118.15 | 142.74 | 154.36 | 171.82 | 191.68 | 304.90 |
| | | 2560 x 10240 | 77.66 | 99.02 | 119.97 | 130.06 | 148.89 | 170.22 | 322.56 |
| | Llama-2-7B | 4096 x 4096 | 53.87 | 68.54 | 80.96 | 88.65 | 98.43 | 116.01 | 206.88 |
| | | 11008 x 4096 | 128.10 | 162.52 | 198.02 | 217.67 | 246.35 | 291.93 | 538.21 |
| | | 4096 x 11008 | 127.33 | 186.92 | 218.29 | 219.58 | 256.50 | 297.60 | 553.73 |
| | Mistral-7B | 14336 x 4096 | 164.20 | 209.12 | 255.67 | 282.39 | 321.54 | 377.20 | 691.33 |
| | | 4096 x 14336 | 158.01 | 206.10 | 252.93 | 284.34 | 328.82 | 359.19 | 717.70 |
| | OPT-6.7B | 16384 x 4096 | 186.25 | 237.60 | 289.68 | 321.12 | 361.60 | 430.34 | 788.32 |
| | | 4096 x 16384 | 179.27 | 236.02 | 288.04 | 312.59 | 352.66 | 404.15 | 805.25 |

*Table 10.* Comparison of matrix-vector multiplication latency against existing uniform quantization kernels. The best results are highlighted for each case.

| GPU | | RTX 4090 | | | RTX 4070 Laptop | | | Jetson AGX Orin | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Matrix Size | | 4096x4096 | 11008x4096 | 4096x11008 | 4096x4096 | 11008x4096 | 4096x11008 | 4096x4096 | 11008x4096 | 4096x11008 |
| 3-bit | ExLlamaV2 | 12.15 | 24.89 | 24.95 | 31.8 | 74.72 | 74.94 | 67.1 | 137.5 | 134.45 |
| | LUT-GEMM | 12.52 | 25.03 | 25.39 | 35.04 | 83.23 | 85.05 | 61.9 | 131.66 | 140.36 |
| | Ours | **9.67** | **21.90** | **23.16** | **28.57** | **71.71** | **72.21** | **53.87** | **128.10** | **127.33** |
| 4-bit | ExLlamaV2 | 14.89 | 31.11 | 31.88 | 41.57 | 97.25 | **97.28** | 81.1 | 171.61 | 172.68 |
| | AWQ | 13.89 | 30.96 | 30.21 | 41.04 | 100.83 | 105.26 | 73.12 | 176.69 | 177.77 |
| | TRT-LLM | **12.58** | **29.34** | **27.94** | 39.58 | 96.86 | 98.98 | - | - | - |
| | LUT-GEMM | 14.78 | 31.73 | 31.78 | 45.35 | 108.33 | 109.87 | 71.64 | 164.27 | **170.49** |
| | Ours | 12.73 | 29.8 | 30.09 | **38.12** | **96.22** | 104.42 | **68.54** | **162.52** | 186.92 |
| 5-bit | ExLlamaV2 | 16.1 | **35.92** | **36.11** | 48.39 | 120.49 | **120.75** | 90.68 | **194.54** | **194.19** |
| | LUT-GEMM | 17.52 | 38.39 | 38.72 | 54.17 | 137.43 | 138.76 | 87.37 | 203.21 | 221.47 |
| | Ours | **14.80** | 36.26 | 37.56 | **47.16** | **120.09** | 125.35 | **80.96** | 198.02 | 218.29 |
| 6-bit | ExLlamaV2 | 19.74 | 43.06 | **43.48** | 58.15 | **147.21** | **147.84** | 101.75 | 228.82 | 223.38 |
| | LUT-GEMM | 19.37 | 45.14 | 46.26 | 66.45 | 165.59 | 166.18 | 100.46 | 236.18 | 251.02 |
| | Ours | **17.72** | **41.95** | 45.19 | **56.55** | 149.99 | 151.54 | **88.65** | **217.67** | **219.58** |
| 7-bit | LUT-GEMM | 22.54 | 51.81 | 51.54 | 73.23 | 191.64 | 192.36 | 117.89 | 273.49 | 301.48 |
| | Ours | **20.64** | **48.13** | **49.55** | **67.48** | **179.50** | **179.48** | **98.43** | **246.35** | **256.50** |
| 8-bit | ExLlamaV2 | **23.52** | **53.07** | **53.55** | **75.79** | **193.00** | **192.91** | 129.96 | 303.93 | **292.41** |
| | LUT-GEMM | 25.04 | 59.27 | 59.01 | 83.61 | 216.59 | 217.3 | 125.12 | 308.45 | 328.92 |
| | Ours | 24.69 | 57.44 | 56.98 | 80.65 | 214.07 | 204.47 | **116.01** | **291.93** | 297.6 |

*Table 11.* Matrix-matrix multiplication latency of our kernel with batch sizes of 2, 4 and 8. We also present the rate of latency increase against single-batch for each case. Cases with latency increases of less than 30% are highlighted in blue, while those which exhibit more than a twofold increase are highlighted in red.

| GPU | | RTX 4090 | | | RTX 4070 Laptop | | | Jetson AGX Orin | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | BS | 4096x4096 | 11008x4096 | 4096x11008 | 4096x4096 | 11008x4096 | 4096x11008 | 4096x4096 | 11008x4096 | 4096x11008 |
| 3-bit | 1 | 9.67 (×1.00) | 21.90 (×1.00) | 23.16 (×1.00) | 28.57 (×1.00) | 71.71 (×1.00) | 72.21 (×1.00) | 53.87 (×1.00) | 128.10 (×1.00) | 127.33 (×1.00) |
| | 2 | 10.73 (×1.11) | 23.77 (×1.09) | 28.36 (×1.22) | 33.72 (×1.18) | 72.87 (×1.02) | 88.15 (×1.22) | 59.81 (×1.11) | 141.49 (×1.10) | 152.23 (×1.20) |
| | 4 | 11.98 (×1.24) | 32.27 (×1.47) | 30.45 (×1.31) | 32.65 (×1.14) | 76.79 (×1.07) | 80.50 (×1.11) | 87.93 (×1.63) | 215.84 (×1.68) | 216.02 (×1.70) |
| | 8 | 15.65 (×1.62) | 44.29 (×2.02) | 40.55 (×1.75) | 51.12 (×1.79) | 121.58 (×1.70) | 125.21 (×1.73) | 151.02 (×2.80) | 388.55 (×3.03) | 378.78 (×2.97) |
| 4-bit | 1 | 12.73 (×1.00) | 29.80 (×1.00) | 30.09 (×1.00) | 38.12 (×1.00) | 96.22 (×1.00) | 104.42 (×1.00) | 68.54 (×1.00) | 162.52 (×1.00) | 186.92 (×1.00) |
| | 2 | 13.93 (×1.09) | 37.92 (×1.27) | 39.12 (×1.30) | 41.38 (×1.09) | 100.14 (×1.04) | 107.57 (×1.03) | 75.91 (×1.11) | 183.80 (×1.13) | 191.78 (×1.03) |
| | 4 | 14.93 (×1.17) | 38.53 (×1.29) | 39.17 (×1.30) | 41.99 (×1.10) | 101.35 (×1.05) | 106.85 (×1.02) | 95.10 (×1.39) | 232.20 (×1.43) | 240.12 (×1.28) |
| | 8 | 17.53 (×1.38) | 48.71 (×1.63) | 45.45 (×1.51) | 57.37 (×1.51) | 135.49 (×1.41) | 140.48 (×1.35) | 175.37 (×2.56) | 452.44 (×2.78) | 430.61 (×2.30) |
| 5-bit | 1 | 14.80 (×1.00) | 36.26 (×1.00) | 37.56 (×1.00) | 47.16 (×1.00) | 120.09 (×1.00) | 125.35 (×1.00) | 80.96 (×1.00) | 198.02 (×1.00) | 218.29 (×1.00) |
| | 2 | 15.40 (×1.04) | 36.33 (×1.00) | 42.67 (×1.14) | 49.26 (×1.04) | 122.81 (×1.02) | 138.92 (×1.11) | 85.92 (×1.06) | 208.13 (×1.05) | 218.24 (×1.00) |
| | 4 | 18.33 (×1.24) | 42.45 (×1.17) | 47.52 (×1.27) | 51.62 (×1.09) | 124.37 (×1.04) | 135.16 (×1.08) | 104.15 (×1.29) | 254.89 (×1.29) | 263.17 (×1.21) |
| | 8 | 22.53 (×1.52) | 50.97 (×1.41) | 54.47 (×1.45) | 61.49 (×1.30) | 134.38 (×1.12) | 152.76 (×1.22) | 136.60 (×1.69) | 334.85 (×1.69) | 446.29 (×2.04) |
| 6-bit | 1 | 17.72 (×1.00) | 41.95 (×1.00) | 45.19 (×1.00) | 56.55 (×1.00) | 149.99 (×1.00) | 151.54 (×1.00) | 88.65 (×1.00) | 217.67 (×1.00) | 219.58 (×1.00) |
| | 2 | 17.89 (×1.01) | 43.41 (×1.03) | 48.21 (×1.07) | 61.90 (×1.09) | 158.64 (×1.06) | 173.25 (×1.14) | 93.35 (×1.05) | 223.61 (×1.03) | 236.13 (×1.08) |
| | 4 | 20.61 (×1.16) | 47.45 (×1.13) | 53.43 (×1.18) | 59.48 (×1.05) | 153.59 (×1.02) | 161.73 (×1.07) | 110.99 (×1.25) | 271.59 (×1.25) | 284.77 (×1.30) |
| | 8 | 25.19 (×1.42) | 55.84 (×1.33) | 60.69 (×1.34) | 70.47 (×1.25) | 165.17 (×1.10) | 177.89 (×1.17) | 148.84 (×1.68) | 362.45 (×1.67) | 502.36 (×2.29) |
| 7-bit | 1 | 20.64 (×1.00) | 48.13 (×1.00) | 49.55 (×1.00) | 67.48 (×1.00) | 179.50 (×1.00) | 179.48 (×1.00) | 98.43 (×1.00) | 246.35 (×1.00) | 256.50 (×1.00) |
| | 2 | 21.74 (×1.05) | 49.79 (×1.03) | 56.18 (×1.13) | 70.92 (×1.05) | 183.65 (×1.02) | 189.63 (×1.06) | 105.25 (×1.07) | 255.30 (×1.04) | 268.83 (×1.05) |
| | 4 | 23.92 (×1.16) | 52.73 (×1.10) | 60.77 (×1.23) | 69.44 (×1.03) | 181.75 (×1.01) | 185.54 (×1.03) | 122.35 (×1.24) | 295.96 (×1.20) | 319.96 (×1.25) |
| | 8 | 28.15 (×1.36) | 59.43 (×1.23) | 67.21 (×1.36) | 79.33 (×1.18) | 193.08 (×1.08) | 205.41 (×1.14) | 157.09 (×1.60) | 384.32 (×1.56) | 517.72 (×2.02) |
| 8-bit | 1 | 24.69 (×1.00) | 57.44 (×1.00) | 56.98 (×1.00) | 80.65 (×1.00) | 214.07 (×1.00) | 204.47 (×1.00) | 116.01 (×1.00) | 291.93 (×1.00) | 297.60 (×1.00) |
| | 2 | 26.64 (×1.08) | 58.35 (×1.02) | 67.43 (×1.18) | 81.81 (×1.01) | 220.69 (×1.03) | 218.08 (×1.07) | 123.00 (×1.06) | 300.90 (×1.03) | 319.13 (×1.07) |
| | 4 | 27.46 (×1.11) | 59.28 (×1.03) | 68.35 (×1.20) | 82.97 (×1.03) | 217.70 (×1.02) | 212.40 (×1.04) | 136.70 (×1.18) | 335.56 (×1.15) | 351.07 (×1.18) |
| | 8 | 31.75 (×1.29) | 66.29 (×1.15) | 71.24 (×1.25) | 90.61 (×1.12) | 224.79 (×1.05) | 225.71 (×1.10) | 172.31 (×1.49) | 416.19 (×1.43) | 529.39 (×1.78) |
| FP16 | | 38.59 | 102.34 | 103.04 | 142.05 | 369.12 | 382.21 | 206.88 | 538.21 | 553.73 |