QUIK: Towards End-to-end 4-Bit Inference on Generative Large Language Models

Saleh Ashkboos[*1] Ilia Markov[*2] Elias Frantar[2] Tingxuan Zhong[3] Xingchen Wang[3] Jie Ren[4] Torsten Hoefler[1] Dan Alistarh[2,5]

[1]ETH Zurich, [2]Institute of Science and Technology Austria,
[3]Xidian University, [4]KAUST, [5]Neural Magic, Inc.

[*]Equal contribution, **Correspondence:** saleh.ashkboos@inf.ethz.ch

Abstract

Large Language Models (LLMs) from the GPT family have become extremely popular, leading to a race towards reducing their inference costs to allow for efficient local computation. However, the vast majority of existing work focuses on weight-only quantization, which can reduce runtime costs in the memory-bound one-token-at-a-time generative setting, but does not address costs in compute-bound scenarios, such as batched inference or prompt processing. In this paper, we address the general quantization problem, where *both weights and activations* should be quantized, which leads to computational improvements in general. We show that the majority of inference computations for large generative models can be performed with both weights and activations being cast to 4 bits, while at the same time maintaining good accuracy. We achieve this via a hybrid quantization strategy called QUIK that compresses most of the weights and activations to 4-bit, while keeping a small fraction of "outlier" weights and activations in higher-precision. QUIK is that it is designed with computational efficiency in mind: we provide GPU kernels matching the QUIK format with highly-efficient layer-wise runtimes, which lead to practical end-to-end throughput improvements of up to 3.4x relative to FP16 execution. We provide detailed studies for models from the OPT, LLaMA-2 and Falcon families, as well as a first instance of accurate inference using quantization plus 2:4 sparsity. Anonymized code is available here.

1 Introduction

Large language models (LLMs) from the Generative Pretrained Transformer (GPT) family (Radford et al., 2019) are extremely popular. One surprising property is the ability to quantize them, e.g., (Frantar et al., 2022; Dettmers et al., 2022; Lin et al., 2023; Yuan et al., 2023), enabling efficient local generative inference for these models, even on personal computers. The vast majority of work on LLM quantization can be categorized as follows:

- *Weight-only quantization methods* (Frantar et al., 2022; Dettmers et al., 2022; Lin et al., 2023; Dettmers et al., 2023; Lin et al., 2023; Kim et al., 2023) that help reduce the massive memory-transfer costs of LLM inference. Yet, these methods do not reduce computation, and cannot provide significant speedup for computationally-bound settings, such as prompt processing.

- *Joint weight-activation quantization methods*, which can provide computational improvements, but either focus exclusively on 8-bit weights and activations (8W8A) (Xiao et al., 2022; Dettmers et al., 2022), or execute with large amounts of accuracy loss relative to their uncompressed counterparts (Yuan et al., 2023; Shao et al., 2023).

Thus, there is still a significant gap between *compressed formats supported by hardware*—specifically, NVIDIA GPUs natively support accelerated 4bit matrix multiplication on both the Ampere and Lovelace architectures (NVIDIA, 2023)—and *quantization algorithms* which would allow *accurate* inference on hardware-supported formats.

Contribution. In this paper, we look to bridge this gap, and show that a large fraction of the computation in modern LLMs such as OPT (Zhang et al., 2022), LLaMA-2 (Touvron et al., 2023) and Falcon (TII UAE, 2023) can be performed accurately and efficiently using *4-bit activations and weights (4W4A)*.

On the algorithmic side, we show significantly improved results relative to prior work on joint quantization of weights and activations to 4 bits, via a hybrid scheme for **QU**antization to **I**NT4 with GPU **K**ernel support, called **QUIK**. In QUIK, matrices are split into "base" weights and activations, which are processed exclusively at 4-bit precision, and a small number of "outlier" weights and activations, which are processed at higher precision such as INT8 or FP16. Using this approach, as well as additional insights into layer sensitivity, we build a

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 3355–3371
November 12-16, 2024

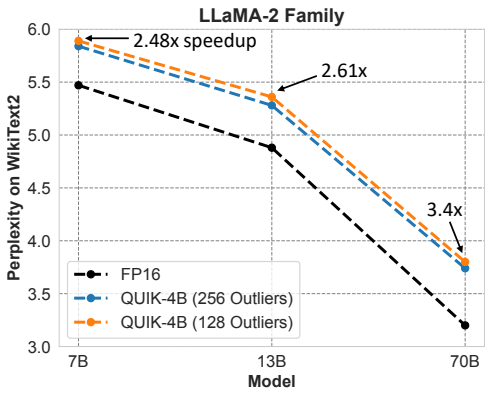


Figure 1: Accuracy and speedups for QUIK at different model sizes, on the LLaMA family of models. QUIK achieves up to 3.4x speedup with minor accuracy degradation on LLaMA-2 models.

framework which can recover accuracy within 0.3–0.5 perplexity points across model sizes, while executing a large fraction of the inference in INT4. For illustration, for the sensitive LLaMA2 model with 70B parameters, we can recover accuracy within 0.5 perplexity, while executing 70% of the linear layer computations in INT4, leading to 3.4x end-to-end speedups (see Figure 1). We consider our work orthogonal to KV-Cache quantization.

On the performance side, the key feature of QUIK is that it can be implemented efficiently via GPU kernels with low runtime and memory overheads relative to GPU-native INT4 matrix multiplication (MatMul). We demonstrate this via a general implementation leading to per-layer speedups and end-to-end throughput improvements relative to both FP16 and INT8 baselines. Specifically, we show that supporting a limited number of feature and weight outliers can have negligible overhead by fusing the quantization and dequantization operations into the MatMul and by mitigating their costs in linear layers via additional optimizations.

Overall, QUIK leverages quantization for significant end-to-end speedups and memory reductions. For example, for processing a sequence of 2048 tokens on a commodity RTX 3090 GPU, we achieve end-to-end speedups between 3.1x, for the OPT-66B and Falcon-180B models, and 3.4x for LLaMA2-70B, relative to a theoretical optimum of ≈4x. In addition, QUIK requires much less GPU memory, and therefore, less GPUs, relative to FP16. For instance, QUIK provides 3.6x memory reduction for OPT-66B, and 3x compression for accurate execution of LLaMA2-70B, executing the latter in less than 50GB of GPU memory.

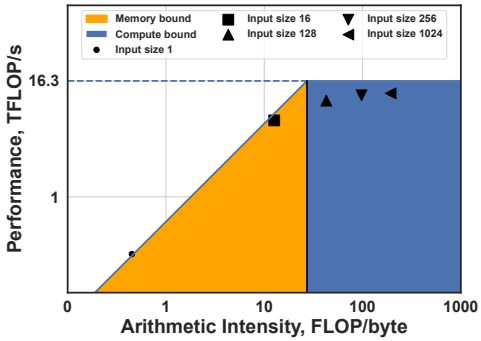


Figure 2: Roofline analysis of a standard LLM MatMul operation, for a matrix of size 8K x 8K, in FP32, on an NVIDIA GPU. Markers denote the results of profiling with different token counts (from 1 to 1024). Small counts (1 and 16) are memory-bound, whereas larger counts (from 128 to 1024) are compute-bound.

2 Motivation

Roofline Analysis. To motivate our focus on the compute-bound case, we begin an analysis of the basic computational operation in the context of LLMs, a matrix multiplication for different numbers of tokens. We profile a linear layer of standard size (11K x 4K, corresponding to the MLP in LLaMA-7B (Touvron et al., 2023)), using the NVIDIA NSight Toolkit (NVIDIA), from a single token to 16, 256 and 1024 tokens.

Figure 2 clearly shows that the case of few tokens (1 and 16) the operation is bound by memory transfer, whereas the it becomes *compute-bound* for token counts larger than 64-128. A realistic end-to-end LLM deployment would need to consider optimizing both scenarios, as the prompt processing "prefill" case falls into the large token count scenario, whereas generating one-token-at-a-time falls into the former case. Moreover, running a "batched" version of the single-token workload, i.e. for multiple users, would again result in large token counts, returning to the compute-bound case.

Notice that existing methods for weight-only quantization (Frantar et al., 2022; Dettmers and Zettlemoyer, 2022; Lin et al., 2023) only reducing the amount of data which needs to be transferred per operation, but still perform the computation in the original precision. Thus, they do not help in the compute-bound case, and in fact even *slightly increase* the amount of computation per operation, due to the de-quantization overheads.

Speedup Potential. Given our focus on the compute-bound case, it is natural to investigate the available hardware options leading to potential speedups. Quantization is a natural approach given that NVIDIA GPUs have native support for INT4

and INT8 data types, providing major throughput improvements across matrix sizes (see Figure 9 in Appendix A). Specifically, INT8 provides throughput improvements that can be slightly higher than 2x relative to FP16, whereas INT4 almost doubles over INT8. However, to leverage these hardware operations, *both layer inputs (activations) and layer weights* must be quantized. We will therefore focus on accurate post-training quantization of accurate pre-trained LLMs, by compressing both weights and activations, primarily to INT4.

3 Method

3.1 Background

We focus on accelerating linear layers within Large Language Models (LLMs) by employing 4-bit quantization for both the weight matrix $\mathbf{W}$ and the input matrix $\mathbf{X}$. Following the PyTorch definition (Paszke et al., 2019), a linear layer with a bias vector $\mathbf{b}$, can be written as $\mathbf{X}\mathbf{W}^\mathbf{T} + \mathbf{b}$. We now describe the technique in detail.

Outliers in Input Quantization. Activation matrices are notoriously hard to quantize accurately (Dettmers et al., 2022; Xiao et al., 2022; Yuan et al., 2023), mainly due to the presence of *outlier features* in these matrices, where some of the columns have up to 100x larger magnitudes. LLM.int8() (Dettmers et al., 2022) identifies and extracts the outlier columns of $\mathbf{X}$ during the forward pass and quantizes the rest of the elements with 8-bit. However, LLM.int8() is not efficient at runtime due to the added computational cost of determining outliers on-the-fly. Recently, Xiao et al. (2022) showed that outlier features are fixed for each layer across datasets, which means that we can extract outlier indices offline using a small calibration set.

Weight Quantization. GPTQ (Frantar et al., 2022) is a weight-only quantization method which involves the quantization of $\mathbf{W}$ while retaining activations $\mathbf{X}$ in FP16. GPTQ iterates over each weight column from left to right, quantizing all column elements simultaneously. Once a certain weight column is quantized, GPTQ adjusts the remaining unquantized columns, to the right, by using second-order information to compensate for the introduced quantization error in the current step. This process naturally *accumulates the quantization errors towards the last columns*.

3.2 QUIK Quantization

Overview. At a high level, QUIK works as follows. First, note that, during the linear transformation $\mathbf{X}\mathbf{W}^\mathbf{T}$, the outlier columns in $\mathbf{X}$, by which we mean the columns with large average values defined previously, will always be multiplied by certain columns in $\mathbf{W}^\mathbf{T}$, as illustrated in Figure 3. We leverage this observation to improve the quality of GPTQ quantization, in a setting where we quantize (part of) the activations as well.

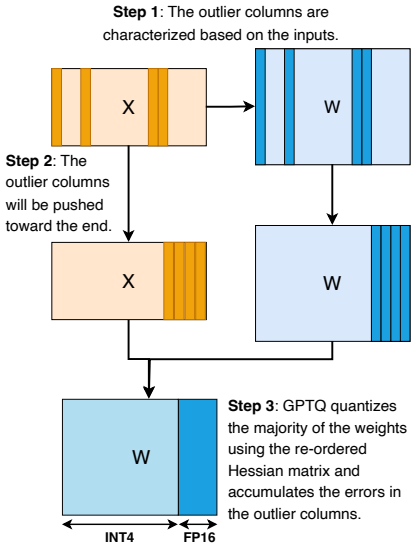


Figure 3: Outlier-aware quantization with QUIK. Outlier weight columns are extracted based on outlier columns in the input. We permute the outlier columns toward the end of the matrix before applying GPTQ quantization (using the re-ordered Hessian matrix) to accumulate the quantization errors in the FP16 columns.

Since the outlier columns are fixed across datasets, we begin by extracting the indices of the outlier columns by means of a calibration set. Then, we rearrange the weight columns (and their corresponding input columns), to shift the outliers toward the end. Finally, we perform quantization on the weight columns up to the index of the outliers. This circumvents quantization of these "difficult" columns. It also helps GPTQ quantization by 1) aggregating the quantization errors to the columns we keep in FP16, and 2) removing potential weight outliers from the 4bit quantization scale.

Sensitivity-Based Partial Quantization. Accurately selecting outlier columns is key for QUIK. Following Xiao et al. (2022); Dettmers et al. (2022), we select the columns with the largest ℓ_∞ norm as outliers. Since finding these columns dynamically at runtime is costly, we follow Xiao et al. (2022) in identifying a predefined set of outliers for each layer via a calibration set (see Section 4), and quantize the weights offline. We use the same outlier indices for extracting the input outlier columns during the forward pass.

This approach is sufficient for accurate quantization of models such as OPT (Zhang et al., 2022) (see Section 4). However, highly-accurate massive models such as LLaMA2-70B present a further challenge due to their FeedForward layers, which involve three linear transformations along with element-wise multiplication, as well as the use of the Sigmoid Linear Unit (SiLU) activations. Specifically, our ℓ_∞ norm analysis illustrated in Figure 11, suggests that the $\text{Down}_{\text{proj}}$ layers are much more sensitive to quantization. (Li et al. (2023) arrived at a similar observation.) Thus, we extend our scheme to improve accuracy by quantizing the $\text{Down}_{\text{proj}}$ layers to 8 bits instead of 4, without other changes to our method. We illustrate the outlier selection procedure in detail in Section 4.3. We present a detailed analysis of our overall FLOP breakdown in Figure 10.

3.3 Efficient GPU Inference

We now provide a high-level description of how models in the QUIK format are executed efficiently on GPU. We illustrate the workflow in Figure 4 and provide detailed pseudocode in Appendix Algorithm 1. The first and most important step in QUIK is splitting the input matrix of shape (#tokens, #features) column-wise, so across features, into two sub-sets, a small "full precision" part (usually half or bfloat16) and a large base part, which will be quantized (see line 2 in the pseudocode). The full-precision part is multiplied with the corresponding (full-precision) part of the weight matrix in standard fashion, while the rest goes through the quantized matrix multiplication pipeline.

The quantized MatMul pipeline consists of three parts: 1) dynamically quantizating the activations, 2) actually performing the MatMul of quantized activations and weights, and 3) dequantizing the result back to floating point format.

Quantization. In general, we quantize weights *symmetrically* (only scale) per column and quantize activations *asymmetrically* (scale and zero) per token. The former is done *offline*, while the latter must be done *online* based on the current activation values. Specifically, we first scan the activations to determine the per-token min- and max-value, from which we calculate the scale and zero point (line 9). These are then used to turn the floating point activations into integers, which are written out again as signed (hence the halfRange subtraction in line 12) INT4 or INT8 values (see lines 10-13).

Matrix Multiplication. The actual MatMul is performed using the NVIDIA `CUTLASS` library (NVIDIA, 2023), which allows us to effectively utilize the hardware's INT8/INT4 tensorcores for fast low-precision calculations, while accumulating results in the INT32 format.

Dequantization. As the MatMul was carried out purely with quantized INT values, we need to convert back to a floating point format in order to properly integrate scale and zero information. Concretely, we need to multiply each output element o_{ij} by its corresponding input token scale `scaleAct` and output weight scale `scaleWeight` (line 16). Additionally, we also need to account for the activation zero-point `zeroAct`. To do this, we consider a scalar product $\langle w, x \rangle$ (representing a single output value in our overall matmul) where a constant z is added to each x_i:

$$y = \sum_i w_i(x_i + z) = \sum_i w_i x_i + z \cdot \sum_i w_i. \quad (1)$$

Consequently, we must shift by z times the *sum over relevant weights*, the latter of which is static and can thus be precomputed as `wReduced`; the signed to unsigined INT conversion must be considered as well (lines 17-21). Finally, we add these dequantized values to the original outlier result, yielding the final output (line 7).

3.4 Performance Optimizations

The main operation in the QUIK kernel is the low-precision `CUTLASS` MatMul. However, the mixed precision nature of the algorithm imposes the use of auxiliary functions, such as input data splitting, metadata computation, quantization and dequantization, which must be carefully optimized.

Quantization Fusion. A naive implementation of splitting and quantization would require one read-and-write pass for the outlier-part, another read-and-write pass for the base-part, two read passes to determine per-token min-max values and one more read-and-write pass for actually carrying out quantization. Many of these slow memory-bound operations can be optimized away via careful operator fusion in the form of bespoke kernels.

Specifically, we assign each input row to a CUDA block and perform 3 passes over it: reduction (finding meta information) over the non-outliers elements, quantization of them, and moving the outliers to a separate piece of memory. This eliminates two costly reads (min-max calculation

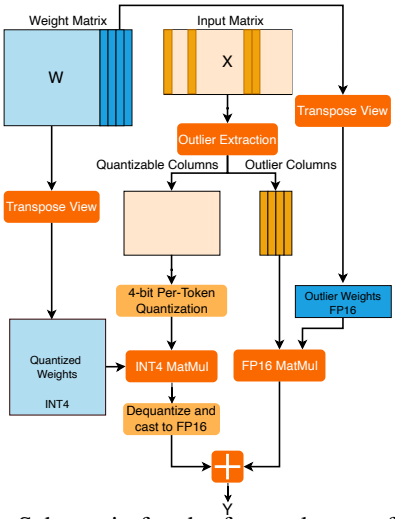


Figure 4: Schematic for the forward pass of a linear layer (XW^T) with QUIK-4B. In the first step, the input outlier features are extracted based on the pre-defined indices and the rest of the input values will be quantized using per-token quantization. The INT4 MatMul will be applied using the quantized weights, calculated offline (see Figure 3). Finally, the output will be dequantized, cast to FP16, and added to the result of FP16 MatMul.

and base-part splitting) and one write pass (base-part splitting), and kernel launches overheads.

Parallelization Tuning. For the above quantization procedure to be efficient on a modern GPU, we have to ensure optimal parallelization via careful tuning of CUDA blocks and threadcounts. The most critical tuning parameter is the number of rows we process with one CUDA block. Mapping one block per each row brings additional launching overheads, while mapping too many rows per block results in block over-subscription and lower occupancy of the GPU. Hence, we optimized the appropriate number of rows per block for different matrix sizes (usually values between 8 and 32). This improved quantization speed by up to 30%.

Dequantization Epilogue. `CUTLASS` first accumulates MatMul results in registers before committing them to global memory. We can avoid an unnecessary write and read pass of intermediate INT32 matmul results by directly performing dequantization in a custom *epilogue* that is applied before the global memory commit, which we further directly accumulate into the results of the outlier MatMul. This interleaves two expensive operations and saves additional kernel calls and memory trips.

Performance Impact. To separate out the impact of these optimizations, we mark them as different versions of our kernel: version 1 has unfused quantization / dequantization; version 2 has fused quantization and unfused dequantization; version 3 fuses both. Figure 5 provides a detailed breakdown of each of these optimizations, showing that they are especially effective for the small matrices, where they lead to end-to-end speedups of almost 2x. Fused quantization gives up to 40% throughput improvement and the dequantization epilogue yields an additional 10% speedup.

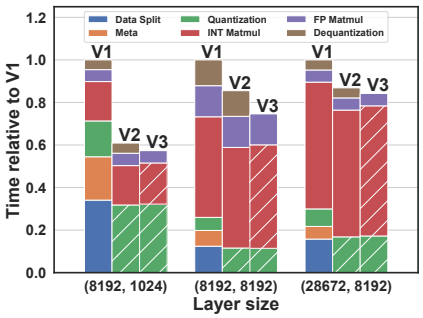


Figure 5: Operation timings in different QUIK-4B versions with 256 outliers on an RTX3090 GPU with input size 2048. Hatched bars represent fused operations.

4 Experimental Validation

General setup. We evaluate our method on OPT (Zhang et al., 2022), LLaMA-2 (Touvron et al., 2023), and Falcon (TII UAE, 2023) models, using HuggingFace (Wolf et al., 2019) implementations of model definitions and datasets. Following SmoothQuant (Xiao et al., 2022), we extract outlier indices using 512 random sentences from the Pile dataset (Gao et al., 2020). We consider up to 5% (based on the model size) of the input features as outliers in the linear layers. During the GPTQ weight quantization, we randomly select 128 samples with 2048 sequence length from the C4 dataset (Raffel et al., 2020). We apply symmetric quantization to weights and asymmetric quantization to activations. Clipping thresholds for weight quantization are found via a linear search over the squared error. QUIK quantizes a 70B model in less than 2h on a single A100 GPU.

4.1 Accuracy Recovery

Accuracy Comparison on OPT. We first compare QUIK with prior 4W4A quantization methods: SmoothQuant (Xiao et al., 2022), RPTQ (Yuan et al., 2023) and OmniQuant (Shao et al., 2023).

Table 1 shows the results of all methods for 4 larger OPT models on the WikiText2 task (Merity et al., 2016). We observed that, with QUIK, the accuracy of OPT models remains consistent even when employing a uniform number of outliers for all layers (instead of using a percentage of the input

Model	OPT			
	6.7B	13B	30B	66B
Baseline	10.86	10.13	9.56	9.34
SmoothQuant	1.8e4	7.4e3	1.2e4	2.2e5
RPTQ	17.83	17.83	11.50	11.16
OmniQuant	12.24	11.65	10.60	10.29
QUIK (ours)	**11.18**	**10.78**	**10.08**	**9.66**

Table 1: Perplexity of 4-bit OPT models on the WikiText2 dataset. SmoothQuant, RPTQ, and OmniQuant results are taken from Shao et al. (2023), RPTQ denotes their improved numbers. For the 66B model, all prior schemes keep 0.71% of the linear layer operations in FP16 (the Head), while, by excluding outliers from quantization, we retain 2.78% of operations in FP16.

features). Consequently, we employed 256 outliers across all linear modules (which is $\approx 3\%$ of OPT-66B's hidden size). As can be seen, by effectively leveraging a small amount of full-precision outlier columns, QUIK can significantly outperform prior 4-bit methods, dropping only 0.3 to 0.5 points in perplexity relative to the full precision baseline. We emphasize that, for a fair comparison, QUIK quantizes *all* linear backbone layers to 4-bit here. Additional results are presented in Appendix I.

Accuracy on LLaMA-2 and Falcon Models. Next, we move to LLaMA-2 and Falcon models. See Table 2 for the results on WikiText2. As can be seen, QUIK-4B can preserve the accuracy in all models with at most 0.5 perplexity loss for the LLaMA-2 models, and 0.3 for Falcon models.

Model	LLaMA-2			Falcon		
	7B	13B	70B	7B	40B	180B
Baseline	5.47	4.88	3.20	6.59	5.23	3.30
SmoothQuant	83.12	35.88	-	-	-	-
OmniQuant	14.26	12.30	-	-	-	-
QUIK-4B	**5.84**	**5.28**	**3.74**	**6.90**	**5.46**	**3.61**

Table 2: Perplexity results of QUIK (with 256 outliers) for 4-bit LLaMA-2 and Falcon models on WikiText2. For the down-projection (in LLaMA-2 models) and FC2 layers (in Falcon models), we use 8-bit quantization, and increase the number of outliers (in FP16) proportionally to the number of input features (which is not the case for other schemes). Results for SmoothQuant and OmniQuant follow (Shao et al., 2023).

Zero-Shot Accuracy. Next, we evaluate the impact of QUIK on the accuracy of zero-shot tasks. To this end, we study the average accuracy of the largest LLaMA-2 and OPT models on five popular zero-shot tasks: PIQA (Tata and Patel, 2003); WinoGrande (Sakaguchi et al., 2021); HellaSwag (Zellers et al., 2019); Arc (Easy and Challenge) (Boratko et al., 2018). We use the LM Evaluation Harness (Gao et al., 2021) with default parameters

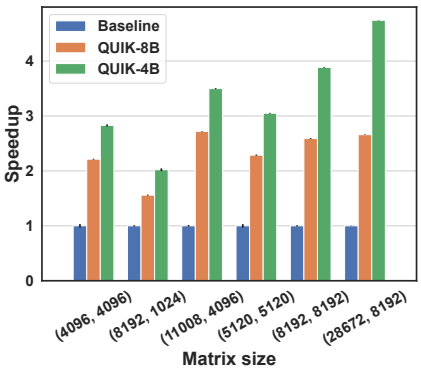


Figure 6: Layer-wise speedups on a single RTX3090 for different layer sizes and compression types. QUIK-4B with 256 outliers, QUIK-8B without outliers.

in our experiments. Table 3 shows the averaged accuracy of QUIK over zero-shot tasks. Similar to the generation task, QUIK preserves the accuracy of zero-shot tasks with at most a 1.5% accuracy drop for LLaMA-2 models and 1.1% for OPT models.

Model	Bits	Avg. Score
OPT-30B	FP16	64.45
	QUIK-4B	63.34
OPT-66B	FP16	66.16
	QUIK-4B	65.10
LLaMA2-13B	FP16	71.70
	QUIK-4B	70.49
LLaMA2-70B	FP16	76.57
	QUIK-4B	74.97

Table 3: LM eval harness results of QUIK on OPT and LLaMA-2 families using 256 outliers. The results are averaged across five different zero-shot tasks. Detailed results are provided in Table 9.

4.2 Performance Analysis

We now examine the performance of the QUIK implementation by evaluating different aspects of our kernel. We use PyTorch/1.13, CUDA/11.8, Huggingface Transformers/4.34. We run all our experiments on RTX 3090 GPUs. Appendix O shows similar results on RTX 3080 GPUs.

Ideal and Layer-wise Speedups. We evaluate the ideal speedups, as well as the actual speedups we measure in each Transformer block separately. The results in Figure 9 depict "ideal" computational power for layer-wise matrix multiplications at different precision levels, without taking into account any quantization/dequantization. Here, we focus on realizable speedups when executing Algorithm 1, which includes mixed-precision multiplication as well as compression and decompression operations.

In Figure 6, we compare the layer-wise performance of quantized linear layers (QUIK-4B uses 256 outliers per layer) relative to FP16, for a full implementation of our algorithm. The matrix sizes correspond to layers in LLaMA models. We observe that QUIK-4B can achieve slightly higher than $4\times$ speedup on large layers and over $2\times$ on smaller ones. Thus, the speedups of raw low-precision matmul speedups can partially "hide" the overheads of QUIK.

End-to-end speedups and Memory Saving. To examine end-to-end speedups, we integrate QUIK into the HuggingFace PyTorch implementation, by replacing linear layers with 4-bit (and 8-bit) QUIK versions. For the LLaMA2 models, we use FlashAttention (Dao et al., 2022) for all models (including FP16). The number of outliers in QUIK-4B is set to 256 except for the special case of down projection layers in LLaMA and FC2 in the Falcon models, which we quantize to 8 bits with $\sim$ 600 outliers. We evaluate memory usage in Appendix C.

In Figure 8, we compare the throughput improvements of prefill passes (for single batches with 2048 tokens) for quantized models, relative to the corresponding FP16 version. The bar plot shows throughput improvements of QUIK-4B compared to FP16. The annotations to the baseline represent its actual throughput values in our experiments. For instance, OPT-66B using FP16 linear layers achieved 439 tokens/s whereas the same model inference with QUIK-4B linear layers resulted in 1343 tokens/s. This shows that, in addition to a close to $4\times$ memory reduction, which reduces the number of required GPUs for inference, QUIK also achieves up to $3.4\times$ higher throughput relative to FP16, with the biggest improvements attained on the largest models (LLaMA2-70B), where the relative impact of overheads is lowest. The memory reduction is important in the Falcon inference case: we were not able to run Falcon-180B in full precision on 8xRTX3090 GPUs, as the max memory peak of the model is more than 360GB. However, QUIK-4B allows us to run full inference of this 180B model on a single server resulting in 542 tokens/second. Therefore, we estimated speedups for the FP16 180B model in Figure 8(c) based on the runtime of a single Transformer block.

The speedups in our end-to-end experiments are exclusively through QUIK accelerated linear layers–other functions are precisely the same. Figure 7 (right) shows that the overheads from attention, softmax, and layernorm operations become significant when most computation occurs in 4-bit.

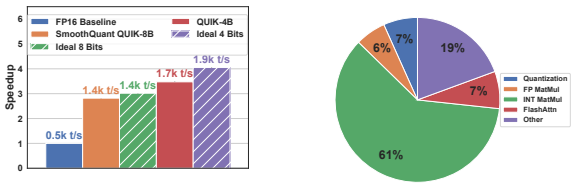


Figure 7: Performance results and overhead breakdown on LLaMA2-70B on a machine with 8x RTX 3090 GPUs. **Left:** Speedup vs. FP16 and vs. an ideal implementation, without overheads, for 4-bit and 8-bit QUIK with absolute throughput values. **Right:** Performance breakdown of end-to-end QUIK inference with outliers in terms of MatMul time vs. quantization overheads.

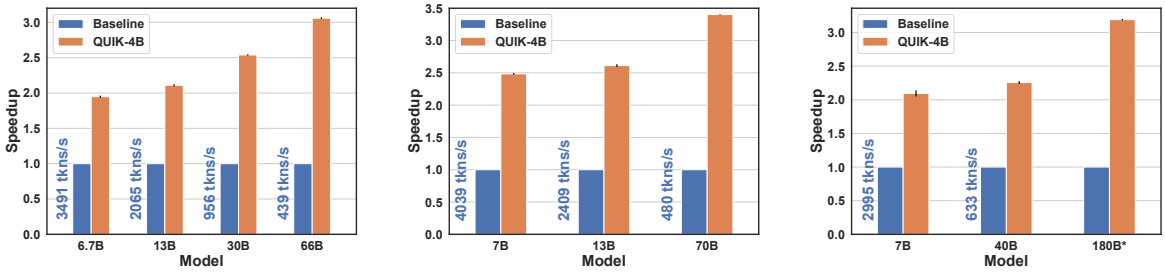
Outlier Performance Costs. To illustrate the overheads of outliers, in Figure 7 (left) we provide end-to-end speedups for variants where we directly use 8-bit and 4-bit kernels, without preserving accuracy (Ideal 8-bit and 4-bit), relative to our accurate QUIK implementations.

We observe that the 8-bit implementation provides close to ideal speedups, reducing the number of GPUs from 7 to 5. QUIK-4B (taking outliers into account) performs $\approx$15% better, further reducing the number of required GPUs to 3, using less than 50 GB of GPU memory. The performance impact of outlier selection (hence mixed precision matrix multiplication) and selective 8-bit quantization (for down-projection MLP layer) is shown in the comparison with Ideal 4-bit. QUIK-4B is within 15% of Ideal 4-bit performance. (Notice that this "Ideal" implementation has very poor accuracy.) In Figure 7 (right) we break down the per-operation overheads for LLaMA2-70B inference. We observe here and in Figure 5 that the overheads of quantization and full precision multiplication can take up a large fraction of the overall operation time, especially for smaller matrices.

4.3 Ablation Studies

We now provide in-depth examples of QUIK on the large LLaMA2-70B and Falcon-180B models. The former model is important as it is highly accurate and sensitive, while the latter is the largest openly-available GPT3-type model.

Case Study 1: LLaMA2-70B. First, we study the FLOP breakdown across precisions using QUIK-4B on LLaMA2-70B. Within the MLP module of the LLaMA2-70B model, three linear layers are present, referred to as "Up-Proj", "Gate-Proj", and "Down-Proj". "Up-Proj" and "Gate-Proj" share



(a) OPT (b) LLaMA-2 (c) Falcon

Figure 8: End-to-end inference speedups for QUIK-4B with outliers relative to the FP16 baseline, on NVIDIA RTX 3090 GPUs. Falcon-180B results are from single Transformer block inference benchmark.

an input (MLP input) and apply their respective linear transformations to it. Subsequently, the output of "Gate-Proj" is subjected to a SiLU activation function. Lastly, the input for the "Down-Proj" layer is constructed by taking the Hadamard product of the outputs from "Up-Proj" and "Gate-Proj".

LLaMA-2	7B	13B	70B
Baseline	5.47	4.88	3.20
QUIK-4B	5.84	5.28	3.74
4-bit Down-Proj	8.87	7.78	6.91

Table 4: Ablation for keeping the $\text{Down}_{\text{proj}}$ in 4-bits.

We use input variance across layers to choose both the number of outliers and the set of layers to be executed in 8bit. (This is illustrated in Figure 11 for LLaMA2-70B.) Specifically, the "Down-Proj" layers have large input variance, mainly due to the Hadamard product of the previous two outputs. To address this, we employ *8-bit quantization* for both the weights and activations within the "Down-Proj" layers of LLaMA2 models. Table 4 shows that keeping the down-projection layers in 8-bit is critical for high accuracy on LLaMA2, as it improves perplexity by > 2 points, across all models.

Case Study 2: Falcon-180B. Finally, we apply QUIK to Falcon-180B, one of the largest GPT-style openly-available models. The model requires $\approx$ 365GB of GPU memory for the inference, which makes it impossible to run inference on a GPU server with 8x RTX3090 nodes (192 GB memory), illustrating the importance of reducing the memory footprint of this model. The results in Tables 2 and 8, and Figure 8 already presented quantization results; in addition we exlore the hardware-supported 2:4 sparse + INT4 format by combining QUIK with 2:4 sparsity.

Instead of just sparsifying the already-quantized model, which results in high accuracy drops, we extend the SparseGPT algorithm (Frantar and Alistarh, 2023) to support our outlier scheme to jointly quantize and sparsify the model, while keeping the outlier features in dense FP16. In Table 5, we present the results of quantizing all layers, but selectively keep some layer types dense. Specifically, we found that one-shot pruning of the weights in the attention blocks to the 2:4 pattern throughout all layers largely preserves accuracy, leading to small memory gains. We present 8-bit results in the same setting in Appendix M.

Precision	Sparsity	Dense Layers	WikiText2 (PPL)	Mem. Peak (rel to FP16)
FP16	0%	All	3.30	100%
	2:4	None	6.13	-
QUIK-4B	0%	All	3.61	38 %
	2:4	None	6.62	25%
	2:4	Attn. Blocks	6.34	26%
	2:4	MLP Blocks	**3.93**	36%

Table 5: Accuracy results for quantized + 2:4 sparsified on Falcon-180B. For the quantized experiments, we apply quantization on all layers with 256 outliers but keep some of the layers in dense (mentioned in the Table) for a single Transformer block.

Discussion. In summary, QUIK shows that one can execute a large majority of inference computation in 4-bit precision, with efficient GPU support. Specifically, one can obtain speedups of over 3x in using QUIK across several LLM types.

5 Limitations

Our current experiments are limited to compressing the linear layers of LLMs. However, our scheme is compatible with virtually any scheme for compressing attention layers or the KV-cache (Sheng et al., 2023), which can be applied orthogonally. Another limitation, which we plan to address in future work, is experimenting with recent Mixture-of-Experts (MoE) architectures, and integration with speculative decoding (Leviathan et al., 2023).

References

Michael Boratko, Harshit Padigela, Divyendra Mikkilineni, Pritish Yuvraj, Rajarshi Das, Andrew McCallum, Maria Chang, Achille Fokoue-Nkoutche, Pavan Kapanipathi, Nicholas Mattei, et al. 2018. A systematic classification of knowledge, reasoning, and context within the ARC dataset. *arXiv preprint arXiv:1806.00358*.

Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and memory-efficient exact attention with io-awareness. *arXiv preprint arXiv:2205.14135*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. LLM.int8(): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022*.

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

Tim Dettmers and Luke Zettlemoyer. 2022. The case for 4-bit precision: k-bit inference scaling laws. *arXiv preprint arXiv:2212.09720*.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot.

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

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

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, et al. 2021. A framework for few-shot language model evaluation. *Version v0. 0.1. Sept*.

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. 2023. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.

Qingyuan Li, Yifan Zhang, Liang Li, Peng Yao, Bo Zhang, Xiangxiang Chu, Yerui Sun, Li Du, and Yuchen Xie. 2023. Fptq: Fine-grained post-training quantization for large language models. *arXiv preprint arXiv:2308.15987*.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. 2023. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*.

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

NVIDIA. Nvidia nsight compute.

NVIDIA. 2023. Nvidia cutlass library.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

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

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2023. Omniquant: Omnidirectionally calibrated quantization for large language models. *Preprint*, arXiv:2308.13137.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pages 31094–31116. PMLR.

Sandeep Tata and Jignesh M Patel. 2003. PiQA: An algebra for querying protein data sets. In *International Conference on Scientific and Statistical Database Management*.

TII UAE. 2023. The Falcon family of large language models. `https://huggingface.co/tiiuae`.

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

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

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

Zhihang Yuan, Lin Niu, Jiawei Liu, Wenyu Liu, Xinggang Wang, Yuzhang Shang, Guangyu Sun, Qiang Wu, Jiaxiang Wu, and Bingzhe Wu. 2023. Rptq: Reorder-based post-training quantization for large language models. *arXiv preprint arXiv:2304.01089*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

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

A Ideal 4-bit Matrix Multiplication

Figure 9 shows the ideal performance of matrix multiplication kernel (using `CUTLASS` library) for different matrix sizes. The plot shows nearly 4x speedup with large enough matrices.

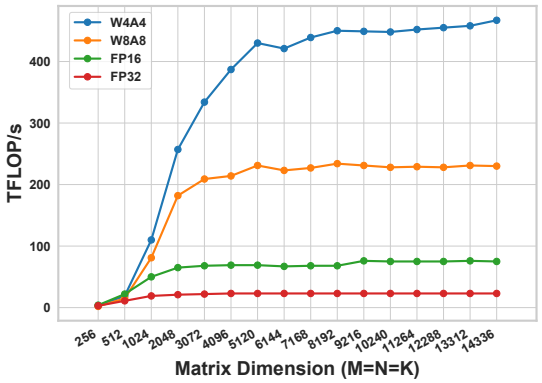


Figure 9: Ideal matrix multiplication performance for different layer sizes and data precision on RTX3090.

B Full QUIK Algorithm

Algorithm 1: Quantization and Dequantization kernels in QUIK.

Input : wInt, wFP, x, FPindices, scaleWeight, wReduced
Function `QUIK Matmul`**:**
  $xFP, xQ \longleftarrow$ split(x, FPindices);
  $xINT, zeroAct, scaleAct \longleftarrow$ Quantization(xQ);
  $resultFP \longleftarrow$ FPmatmul(xFP, wFP);
  $resultInt \longleftarrow$ INTmatmul(xINT, wInt);
  $dequantFP \longleftarrow$ Dequantization(resultInt, zeroAct, scaleAct, scaleWeight, wReduced);
  return dequantFP + resultFP;

Input : dataFP
Function `Quantization`**:**
  $zeroAct, scaleAct \longleftarrow$ findZeroScale(dataFP);
  for *elem* $\in$ *dataFP, outElem* $\in$ *output* **do**
    //Use scale/zero corresponding to token
    $outFP \longleftarrow$ (elem - zeroAct) / scaleAct - halfRange;
    $outElem \longleftarrow$ pack(outFP);
  return output, zeroAct, scaleAct;

Input : inputINT, zeroAct, scaleAct, scaleWeight, wReduced
Function `Dequantization`**:**
  for *elem* $\in$ *inputINT, outElem* $\in$ *outputFP* **do**
    //Use scales for token and weight row, respectively
    $x \longleftarrow$ elem * scaleAct * scaleWeight;
    $shift \longleftarrow$ zeroAct + halfRange * scaleAct;
    $shift \longleftarrow$ shift * wReduced;
    $outElem \longleftarrow x + shift$;
  return outputFP;

C QUIK Peak Memory Usage

In this section, we assess the memory usage of our quantized models. In Table 6, we evaluate the peak memory usage across different configurations for the OPT and LLaMA-2 families. For OPT-66B, the QUIK-8B and QUIK-4B models demonstrate peak memory reductions of approximately 47% (compared to the ideal 50% reduction) and 74% (compared to the ideal 75% reduction), respectively. For the LLaMA2-70B model, the reductions are 32% for QUIK-8B and 67% for QUIK-4B. This is because we keep the down-projection in 8-bits and use additional outliers. Additional overheads come from auxiliary buffers, which differ for various layer sizes.

Model	OPT			LLaMA-2		
	13B	30B	66B	7B	13B	70B
Baseline	30.5	67.4	162.1	14.9	28.0	147.1
QUIK-8B	16.1	39.3	81.2	14.6	25.2	99.3
QUIK-4B	**10.7**	**24.6**	**45.1**	**7.1**	**12.1**	**49.1**

Table 6: Peak memory usage (in GB) in an end-to-end benchmark. In total, the outliers take 2.71 GB and 4.06 GB for OPT-66B and LLaMA2-70B models respectively.

D QUIK FLOP/s Analysis

Figure 10 shows the percentage of the FLOP/s we keep in each precision (INT4 for base weights, FP16 for outliers, and INT8 for down-projection layers) in LLaMA2-70B. More precisely, for 256 outliers, we perform ≈70% of the operations in 4-bit and ≈27% using 8-bits.

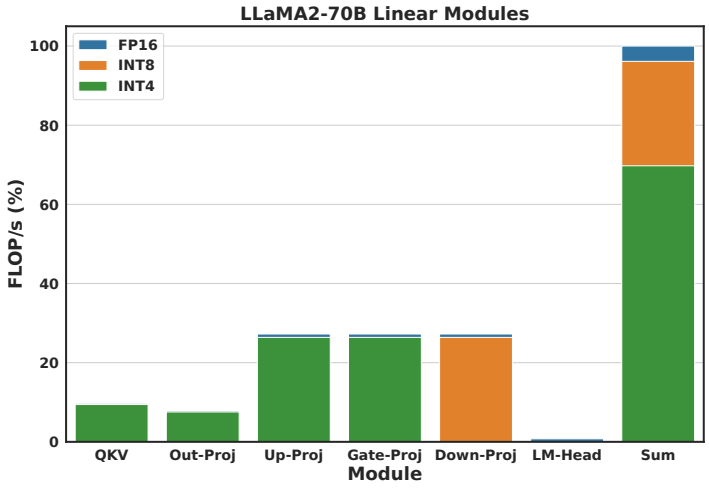


Figure 10: FLOP/s analysis of the LLaMA2-70B linear layers with QUIK. We use 3.125% outliers (256 outliers in all layers and 896 for the down-projection layer) and 2048 sequence length.

E Input Variance of Linear Layers

Figure 11 shows the variance of the inputs for different layers of 70B model.

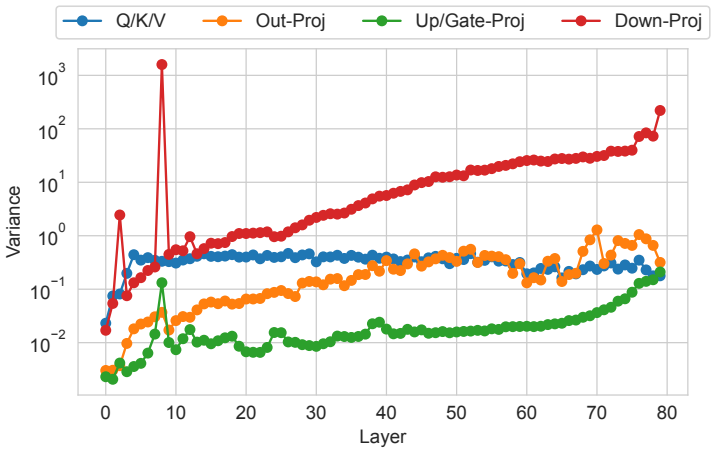


Figure 11: The variance of the inputs in different layers of LLaMA2-70B. The "Down-Proj" layers have significantly larger variances, resulting in poor 4-bit quantization.

F Outlier Analysis

In this section, we look at how different outlier counts affect the WikiText2 score for the LLaMA2-70B model. In Table 7, we observe that increasing the outliers from 128 to 1024 results in a 0.2 perplexity improvement. We also adjusted the outliers for down-projection layers, ensuring there are 3.5x times more than the other linear layers, to match input size. Our results show that using 256 outliers is already a good choice for our experiments. Using additional outliers does not significantly improve accuracy.

Method	Outliers	Down-Proj Outliers	WikiText2 (PPL)
Baseline	-	-	3.20
QUIK-4B	128	448	3.80
	256	896	3.74
	512	1792	3.67
	1024	3584	3.62

Table 7: Ablation study of different outlier numbers in QUIK for the LLaMA2-70B model.

G Outlier-Free Layers

We study the effect of keeping multiple linear layers without any outliers. This might help boost end-to-end performance by removing all the outlier-related overheads during the forward pass. (Although, as we show later, these overheads are minor.) Table 8 shows how the accuracy of different models changes when we use different absolute threshold values (shown by **T**), extracted using a linear search, for the outliers. We conclude that there is no universal threshold across all models, which would preserve accuracy across all models. For example, Falcon-180B can achieve reasonable accuracy even if 24% of the linear layers (115 out of 480) contain zero outliers. However, this is not the case for smaller models: LLaMA2-70B can recover accuracy with up to 5% of the linear layers (30 out of 560) having zero QUIK outliers. We provide additional experiments in Appendix L.

Model	**T**	LLaMA2-70B	Falcon-180B
FP16	-	3.2	3.30
QUIK-4B	0	3.74 (0)	3.61 (0)
	2.0	3.75 (10)	3.61 (3)
	3.0	3.85 (30)	3.61 (4)
	4.0	5.15 (58)	3.72 (14)
	8.0	5.92 (219)	3.73 (115)

Table 8: Study of zero outlier setting on WikiText2 using 256 outliers. We use zero outliers when the maximum of scale is less than threshold **T**. For each experiment, the number of linear layers with zero outliers is written in parentheses.

H Detailed Zero-Shot Results

Table 9 shows the detailed results of QUIK-4B on OPT and LLaMa-2 families on five popular zero-shot tasks: PIQA (Tata and Patel, 2003); WinoGrande (Sakaguchi et al., 2021); HellaSwag (Zellers et al., 2019); Arc (Easy and Challenge) (Boratko et al., 2018). We use the LM Evaluation Harness (Gao et al., 2021) with default parameters in our experiments.

Model	Bits	Arc Challenge	Arc Easy	HellaSwag	PIQA	WinoGrande	Avg. Score
OPT-30B	FP16	38.05	65.36	72.28	78.13	68.43	64.45
	QUIK-4B	36.69	64.39	70.84	77.75	67.01	63.34
OPT-66B	FP16	40.02	67.26	74.87	79.82	68.82	66.16
	QUIK-4B	38.82	64.73	73.68	79.43	68.82	65.10
LLaMA2-13B	FP16	48.98	77.44	79.38	80.52	72.22	71.70
	QUIK-4B	48.04	74.92	78.36	79.22	71.90	70.49
LLaMA2-70B	FP16	57.34	80.98	83.81	82.75	77.98	76.57
	QUIK-4B	56.14	79.00	81.57	81.56	76.56	74.97

Table 9: LM eval harness results of QUIK on OPT andLLaMA-2 families, using 256 outliers.

I Full OPT Accuracy Results

Table 10 shows the perplexity results of OPT models. We use symmetric quantization for the weights in all our experiments. The results suggest that in a 4-bit setting, considering outlier features is crucial to preserve the accuracy even in small models (like OPT-1.3b). We note that 256 outliers is equivalent to 12.5% of the 1.3B model's hidden size (and 2.77% of the 66B model's hidden size).

Model	OPT-1.3b			OPT-6.7b			OPT-13b			OPT-30b			OPT-66b		
Task	WIKI	PT	C4	WIKI	PT	C4	WIKI	PT	C4	WIKI	PT	C4	WIKI	PT	C4
Baseline	14.63	16.96	14.72	10.86	13.09	11.74	10.13	12.34	11.20	9.56	11.84	10.69	9.34	11.36	10.28
GPTQ-4B	15.89	18.83	15.90	11.43	13.81	12.21	10.38	12.65	11.41	9.60	12.02	10.83	9.65	11.63	10.56
0 Outliers	15k	9k	10k	10k	9k	9k	9k	12k	9k	12k	13k	17k	12k	13k	10k
64 Outliers	26.259	27.143	22.981	11.473	13.888	12.348	11.031	13.305	11.971	10.283	12.557	11.267	9.851	11.965	10.742
128 Outliers	17.638	19.709	16.799	11.671	13.809	12.314	10.964	13.241	11.894	10.339	12.564	11.279	9.805	11.842	10.653
256 Outliers	17.358	19.525	16.607	11.184	13.811	12.262	10.779	13.175	11.847	10.078	12.465	11.226	9.662	11.793	10.635

Table 10: Perplexity scores of QUIK-4B over various OPT models with different outliers on three datasets: WikiText2 (WIKI), Pen Treebank (PT), and C4. GPTQ-4B only quantizes the weights (using int-4 symmetric quantization) and keeps the activations in FP16.

J Full LLaMA-2 Accuracy Results

Table 11 shows the perplexity of QUIK on LLaMA-2 models. We provide a list of tricks to improve the quality of the model without too much overhead. We found that keeping the down-proj layer in 8 bits can improve the perplexity by about 3 points. Also, we found weight clipping as a cheap and efficient trick for improving the accuracy of QUIK-4B.

LLaMA-2	Down-Proj	Clipping	7B	13B	70B
FP16	W16A16	-	5.47	4.88	3.2
GPTQ-4B	W4A16	-	6.24	5.25	3.68
QUIK-4B	W4A4	-	8.78	7.78	6.91
QUIK-4B	W4A16	-	6.09	5.49	3.98
QUIK-4B	W4A8	-	6.11	5.5	4.0
QUIK-4B	W8A8	-	5.98	5.37	3.87
QUIK-4B	W8A8	✓	5.84	5.28	3.74

Table 11: LLaMA-2 perplexity results on WikiText2 using 256 outliers. We apply clipping only during the weight quantization.

K Full INT-8 Accuracy Results

Table 12 shows QUIK-8B comparison against SmoothQuant on the WikiText2 dataset. We use per-token (per-column) quantization for the activations (weights) in SmoothQuant and only apply the quantization on the linear layers (which is the case for QUIK also). We exclude the Falcon-7B model as this model has a single layer-norm for both MLP and Attention blocks and it is not clear how the weights of the FC1 and KQV will be updated in the SmoothQuant algorithm.

Model	OPT					LLaMA-2			Falcon	
	1.3b	6.7B	13B	30B	66B	7B	13B	70B	40B	180B
FP16	14.63	10.84	10.13	9.56	9.34	5.47	4.88	3.20	5.23	3.30
SmoothQuant	14.70	10.89	10.37	9.59	9.80	5.58	4.94	3.48	5.26	**3.30**
QUIK-8B	**14.62**	**10.84**	**10.13**	**9.51**	**9.29**	**5.48**	**4.89**	**3.33**	**5.23**	3.31

Table 12: Accuracy results for 8bit models on WikiText2. We use 256 outliers in QUIK experiments. Following the SmoothQuant paper, we use $\alpha = 0.8$ hyperparameter for LLaMA-2 models and $\alpha = 0.5$ for OPT and Falcon families.

L Zero-Outlier Full Results

Table 13 shows the results of keeping different numbers of layers without outliers for different models.

M 2:4 Sparsity + INT8 Quantization

Table 14 shows the accuracy results of applying QUIK-8B with 2:4 sparsity across all models. The results suggest that the main accuracy drop is from introducing 2:4 sparsity to the weight matrices and keeping some of the layers in dense is crucial to preserve the accuracy (See section 4.3).

N Falcon performance benchmark

We also explore the performance improvements of Falcon (TII UAE, 2023) models. The 8xRTX3090 machine contains around 190GB GPU memory which is not enough to run fp16 model inference.

Model	T	LLaMA-2			Falcon		
		7B	13B	70B	7B	40B	180B
FP16	-	5.47	4.88	3.2	6.59	5.23	3.30
QUIK-4B	0	5.84 (0)	5.28 (0)	3.74 (0)	6.90 (0)	5.46 (0)	3.61 (0)
	2.0	5.91 (5)	5.33 (3)	3.75 (10)	6.90 (3)	5.46 (1)	3.61 (3)
	3.0	6.09 (11)	5.34 (8)	3.85 (30)	6.91 (14)	5.46 (2)	3.61 (4)
	4.0	6.13 (21)	5.36 (17)	5.15 (58)	6.93 (27)	10.56 (8)	3.72 (14)
	8.0	12.93 (55)	21.85 (66)	5.92 (219)	6.94 (57)	10.61 (33)	3.73 (115)

Table 13: Study of zero outlier setting on WikiText2 using 256 outliers. We use zero outliers when the maximum of scale is less than threshold **T**. For each experiment, the number of linear layers with zero outliers is written in parentheses.

Model	Sparsity	OPT					LLaMA-2			Falcon		
		1.3b	6.7B	13B	30B	66B	7B	13B	70B	7B	40B	180B
FP16	0%	14.63	10.84	10.13	9.56	9.34	5.47	4.88	3.20	6.59	5.23	3.30
SparseGPT	2:4	24.08	14.15	12.93	10.93	10.08	10.97	8.78	5.70	12.33	12.33	6.13
QUIK-8B	0%	14.62	10.84	10.13	9.51	9.29	5.48	4.89	3.33	6.59	5.23	3.31
	2:4	22.69	14.59	12.87	11.06	10.24	11.07	8.66	5.89	11.07	8.09	6.19

Table 14: WikiText2 accuracy results for applying 2:4 sparsity with QUIK-8B. We use 256 outliers in all experiments.

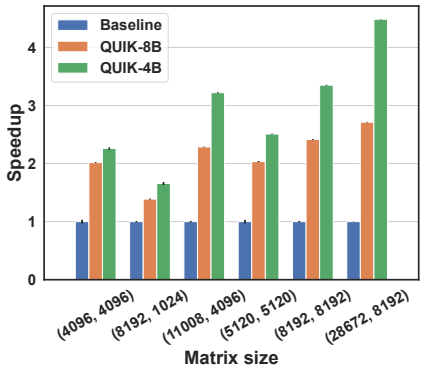


Figure 12: Layer-wise speedups on a single RTX3080 for different layer sizes and compression types. QUIK-4B with 256 outliers, QUIK-8B without outliers.

O Performance on RTX3080 GPUs

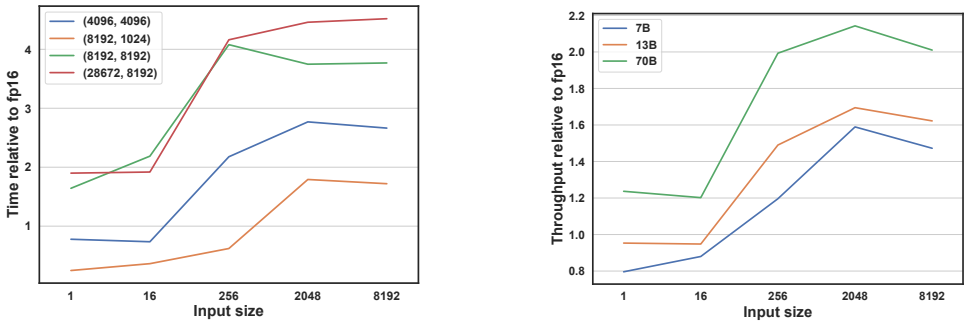
To validate the performance of QUIK in other types of GPUs we conducted benchmarks on RTX3080 GPUs. The results are presented in Figure 12. We can see that QUIK-4B still can get more that 4x speedup on another type of GPU.

P Performance at different sequence sizes

We mainly focus our work on the "prefill" cases with large sequence sizes (in all our experiments sequence size is equal to 2048). In this section we explore the performance of the QUIK-4B with other input sequence sizes. In Figures 13(a) and 13(b) we vary input size from 1 to 8k. In the first expeeriment (Figure. 13(a)) we ran layer-wise benchmark, in the second (Figure 13(b)) we ran inference of a single Transformer block (on a single GPU). We see that at small input sequence sizes QUIK is noticably slower for smaller layer size and models. It can be explained by the fact that the gains of low precision matrix multiplication at this scale can not compensate the quantization overheads. However, at large layer and model sizes QUIK has up to 2x speedup even with single token input. In case of the large input sequences we see that performance decreases meaning that low precision matrix multiplication saturates at this scale.

Q Performance with various outlier number

In this section we explore the effect of outliers numbers on the QUIK performances. Figure 14 suggests that the timing of QUIK matmul stays the same across all layer sizes for all non-zero outlier numbers. The zero outliers case superiority can be explained by the fact that it does not have additional full precision matrix multiplication and input data movements. However, these results show that QUIK allow increase the outlier number without performance sacrifices which is crucial for the accuracy recovery, as we discussed in the Section **??**.



(a) Layerwise Performance. (b) LLaMA Block performance.

Figure 13: Relative performance of QUIK-4B with outliers for different sequence sizes (batch size = 1) on RTX3090 GPU

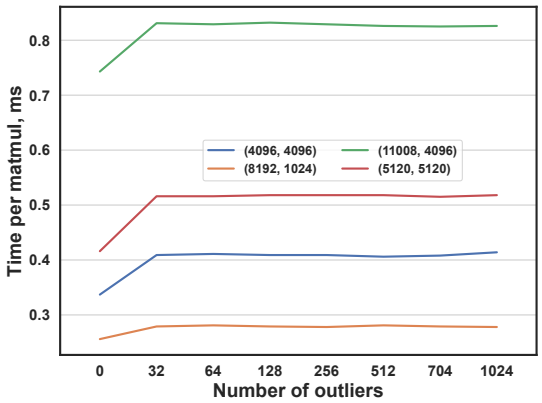


Figure 14: Timing results for different QUIK-4B layers sizes with various number of outliers on RTX3090 GPU.