

Figure 4: Comparison of various scaling law fits and their errors.

## A Technical Appendices and Supplementary Material

### A.1 Training Hyper-parameters

Table 4 lists model-specific hyper-parameters. Table 5 lists hyper-parameters shared across all experiments.

Hyperparameter	30M	50M	100M	200M	7B
Number of Layers ( $N_{\text{layer}}$ )	6	7	8	10	32
Embedding Dimension ( $N_{\text{embd}}$ )	640	768	1024	1280	4096
Attention Heads ( $N_{\text{head}}$ )	5	6	8	10	32
Learning Rate (LR)	0.0012	0.0012	0.0006	0.0003	$9.375 \cdot 10^{-6}$

Table 4: Model-specific hyperparameters used in our experiments.

Hyperparameter	Value
Sequence Length	512
Batch Size	512
Optimizer	AdamW
Learning Rate Schedule	Cosine decay with 10% warm-up
Gradient Clipping	1.0
Weight Decay ( $\gamma$ )	0.1
Number of GPUs	8
Data Type (optimizer/accumulators)	FP32

Table 5: Common hyperparameters used across all model sizes and quantization setups.

### A.2 Scaling Law fitting

We fit the scaling law in two stages:

**Stage 1.** Identical to prior work [8], we fit the unquantized scaling law of the form

$$L(N, D) = \left( \frac{A}{N^\alpha} + \frac{B}{D^\beta} \right)^\gamma + E$$

on baseline BF16 runs for  $N \in [30M, 50M, 100M, 200M]$  and  $D/N \in [25, 50, 100, 200, 400, 800]$  (see Figure 1(a)) using Huber loss with  $\delta = 10^{-4}$  on logarithm of  $L$ . Table 6 shows the resulting fit.

**Stage 2.** Using the fixed fitted parameters from **stage 1**, we fit the additional  $\text{eff}_N$  and  $\text{eff}_D$  parameters using the same loss function.

For the isolated methods compared in Section 4.2, we fit  $\text{eff}_N$  and  $\text{eff}_D$  independently for forward-only and backward-only quantization respectively.

For the end-to-end 4-bit comparison in Section 5, we fitted the parameters jointly for the setups present in Table 3.

**Alternative forms.** We additionally for the scaling law forms with fixed  $\gamma = 1$  [24] and  $\beta = 1$  [25]. The fits are presented in Figure 4 alongside the mainly used of Busbridge et al. [8].

Parameter	$A$	$\alpha$	$B$	$\beta$	$E$	$\gamma$
Value	$1.52 \cdot 10^5$	0.589	$5.25 \cdot 10^5$	0.544	1.35	0.274

Table 6: Fitted scaling law coefficients.

### A.3 Performance breakdown

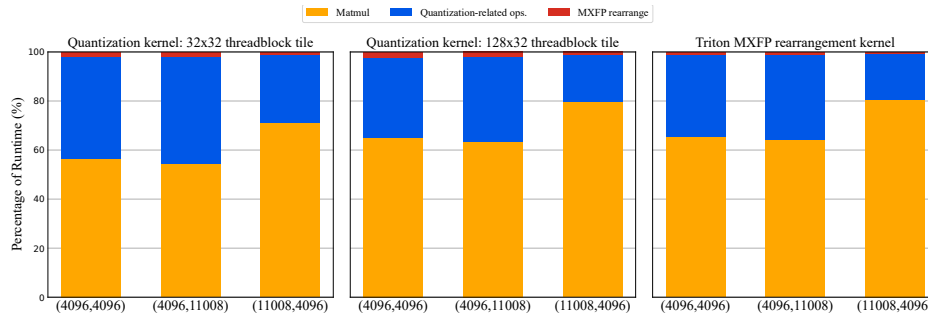


Figure 5: Breakdown of runtime composition across three linear layer shapes of a Llama-7B model, for an input of batch size 64, and sequence length 512.

Figure 5 presents a breakdown of runtime composition across three linear layer shapes in a LLaMA-7B model, taking the MXFP4 forward pass as an example. Each subplot shows the percentage of total runtime spent in three key kernel stages: matrix multiplication, quantization-related operations, and rearrangement of scaling factors for `tcgen05.mma` [31].

The figure compares three kernel configurations. The left subplot shows our fused kernel for quantization-related operations using a basic  $32 \times 32$  threadblock tile size. The center subplot increases this tile size to  $128 \times 32$ , resulting in a more efficient quantization stage. The right subplot includes a custom Triton kernel, which further improves performance by optimizing the MXFP rearrangement stage. All results are normalized to 100%.

As the figure illustrates, tuning the quantization kernel significantly reduces the proportion of time spent in the quantization stage—particularly for large matrix shapes. Increasing the threadblock tile size leads to more active warps per block, enhancing arithmetic intensity and enabling better latency hiding. In CUTLASS-based implementations, this change influences the multilevel tiling strategy (threadblock, warp, and instruction-level tiling), which is designed to optimize data movement through shared memory and registers [39]. The Triton backend exhibits similar trends, with rearrangement overheads further reduced and matrix multiplication dominating the total runtime.

### A.4 End-to-end prefill speedups

Figure 6 illustrates the inference prefill speedup of MXFP4 over FP8 as a function of batch size, evaluated at a fixed sequence length of 256 on a 7B parameter model. The results demonstrate a consistent improvement in performance using MXFP4 across all batch sizes, with speedup increasing progressively and peaking at  $1.41\times$  relative to FP8 at a batch size of 128, where it plateaus.

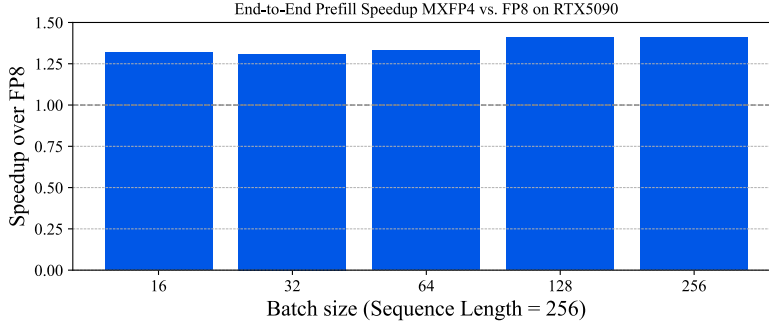


Figure 6: End-to-end prefill speedups for Quartet MXFP4 vs. FP8, across different batch sizes, using the 7B parameter model on a single RTX 5090.

## A.5 Post-Training Quantization Results

We compare the results of applying post-training quantization (PTQ) against QUARTET using the MXFP4 format on the largest 7B model. For the PTQ baseline, we evaluate against QUAROT [2], where the weights are quantized using GPTQ [20]. To ensure a fair comparison, we introduce two key modifications to the original QUAROT approach:

- Attention Module:** We remove the use of online Hadamard transformations and instead apply a fixed Hadamard transformation of size 128 to the output dimension of the  $v\_proj$  layer and the input dimension of the  $out\_proj$  layer. This optimization accelerates the overall process by eliminating per-head online Hadamard computations, without affecting accuracy, since we use a group size of 32 in the MXFP4 format.
- MLP Down-Projection:** For  $down\_projection$  layers with non-power-of-two dimensions in the MLP, we apply grouped Hadamard transformations using the largest power-of-two size that evenly divides the intermediate dimension of the MLP.

Model Size	BF16	QuaRot (PTQ)	Quartet
7B	16.40	18.19	17.77

Table 7: Perplexity results on C4 dataset using MXFP4 quantization. We use 128 samples from the training set (of the same dataset) as the calibration set in GPTQ.

Table 7 presents the comparison between the PTQ scheme (QuaRot) and QUARTET. QUARTET achieves a 0.42-point lower perplexity (PPL) compared to QuaRot when applied to the same model. Notably, QUARTET is also more efficient than standard QAT methods, as it quantizes both forward and backward passes.

## A.6 Compute Resources

The pre-training experiments were conducted on datacenter-grade machines with 8xH100 NVIDIA GPUs for a total compute of around 6,000 GPU-hours. Although most experiments do not require such an elaborate setup, we found the 7B pre-training experiment specifically to be very DRAM-demanding and requiring such specific hardware.

The speedup results were obtained on a consumer-grade NVIDIA RTX5090 GPU with total runtime of under 1 hour.