Quartet: Native FP4 Training Can Be Optimal for Large Language Models

Roberto L. Castro [*1] **Andrei Panferov** [*1] **Rush Tabesh** [1] **Jiale Chen** [1]
Oliver Sieberling [2] **Mahdi Nikdan** [1] **Saleh Ashkboos** [2] **Dan Alistarh** [1 3]

Abstract

Training large language models (LLMs) models directly in low-precision offers a way to address computational costs by improving both throughput and energy. NVIDIA's recent Blackwell architecture facilitates very low-precision operations using FP4 variants for efficiency gains. Yet, current algorithms for training LLMs in FP4 precision face significant accuracy degradation and often rely on mixed-precision fallbacks. In this paper, we investigate hardware-supported FP4 training and introduce Quartet, a new approach for accurate, end-to-end FP4 training with all the major computations (in e.g. linear layers) in low precision. Through extensive evaluations on Llama-type models, we reveal a new low-precision scaling law that quantifies performance trade-offs across bit-widths and training setups. Guided by it, we design an optimal technique in terms of accuracy-vs-computation, called Quartet. We implement Quartet using optimized CUDA kernels tailored for Blackwell, demonstrating that fully FP4-based training is a competitive alternative to standard-precision and FP8 training.

Introduction

One key lever for reducing the massive compute costs of LLMs is *lower-precision computation*: executing the matrix-multiplication (MatMul) kernels that dominate training workloads at lower bit-widths yields near-linear gains in throughput and energy efficiency. On the inference side, it is known that 4-bit quantization—or even lower—can preserve accuracy, via sophisticated calibration and rotation schemes [19; 2; 9]. For training, recent work has pushed the precision frontier from FP16 [29] to 8-bit pipelines, responsible in part for efficiency breakthroughs such as DeepSeek-V3 [27]. In this context, NVIDIA's Blackwell architecture introduces efficient hardware support for even lower-precision microscaling formats [31] such as MXFP and NVFP, which natively support 4-bit floating-point operations at higher teraFLOP-per-watt efficiency: for instance, moving from 8- to 4-bit multiplies on the B200 GPU can almost *double* arithmetic throughput, while cutting energy roughly in half [30].

Yet, today's algorithmic support for *accurate end-to-end* training in such low precision is missing. State-of-the-art quantized training methods such as Switchback [46], Jetfire [49], HALO [3], and INT4-Transformers [48] either (i) lose precision and stability when training current models in 4-bit formats, or (2) fall back to higher precision for selected matrix multiplications. Bridging this gap calls for both a deeper understanding of quantization error during back-propagation and new algorithmic safeguards tailored to hardware-native FP4 formats.

Contributions. In this paper, we address this challenge via a first systematic study of hardware-supported FP4 training, focusing on the high-efficiency of the MXFP4 format [31; 30]. Based on this analysis, we introduce an algorithm for MXFP4 native training—in which all matrix multiplications occur in MXFP4—called **Quartet**, which provides the best accuracy-efficiency trade-off among existing methods, and is near-lossless for LLM pre-training in the large-data regime. Our main technical contribution is a highly-efficient GPU implementation of Quartet, which achieves speedups of almost 2x relative to FP8 for linear layer computations on an NVIDIA Blackwell RTX 5090 GPU, relative to a well-optimized FP8 baseline. One key achievement is that Quartet enables MXFP4 precision to be "optimal" on the accuracy-efficiency trade-off: at a fixed computational budget, the accuracy impact of lower-precision training in Quartet is fully compensated by the higher efficiency of our implementation. Specifically, we show for the first time that the new MXFP4 format can be competitive with FP8 in terms of accuracy-vs-speed, which we hope can enable significant reductions in the rising computational costs of AI.

1. Ingredient 1: Comparing Training Approaches via their Induced Scaling Laws

The ability of LLMs to scale predictably with both model size and data across orders of magnitude is a cornerstone of

[*]Equal contribution [1]ISTA [2]ETH Zürich [3]Red Hat AI. Correspondence to: Dan Alistarh <dan.alistarh@ist.ac.at>.

the current AI scaling landscape [24]. Mathematically, this says that the expected loss is a function of model and data parameters, often described in the form of parametric function. This function can be fitted on a set of training runs, and then used to determine the optimal computational training regime [23] or to extrapolate model performance [22].

We investigate scaling laws relating evaluation loss to the precision in which the forward and backward passes are performed, denoted by P_{fw} and P_{bw}, respectively. For this, we propose a scaling law for the loss $L(N, D, P_{fw}, P_{bw})$ of the following functional form:

$$\left(\frac{A}{(N \cdot \text{eff}_N(P_{fw}))^\alpha} + \frac{B}{(D \cdot \text{eff}_D(P_{bw}))^\beta}\right)^\gamma + E, \quad (1)$$

where $A, B, \alpha, \beta, \gamma$ are constants describing the general loss scaling w.r.t. model parameters N and data size D.

The key addition is given by the fitted parameters $\text{eff}_N(P_{fw})$, representing the *parameter efficiency* of the precision P_{fw} used in the forward pass, and $\text{eff}_D(P_{bw})$ representing the "data efficiency" of the backward pass. The former follows the general trend of modeling the effect of forward pass quantization as a multiplicative factor on parameter count [20; 26; 21; 32]. For the latter, we postulate that lowering backward-pass precision primarily impacts the data term D, so we effectively need additional data to reach the same the same loss, precisely by a factor of $1/\text{eff}_D(P_{bw})$. This is a novel way to model backward pass quantization that we propose, consistent with optimization theory results [1], as well as observed performance gaps (see Figure 1 (a)). We compare against alternatives [24; 23] in the Appendix.

Experimentally, we observe that different quantized training methods, e.g. STE [6] vs. QuEST [32], induce different scaling laws, and in particular different efficiency parameters. While, usually, scaling laws are used to extrapolate *model performance* across different parameter and data sizes, *we propose to use scaling laws to compare different training methods*. Specifically, we say that quantized training method A is superior to method B if it offers both higher parameter efficiency eff_N and higher data efficiency eff_D.

2. Ingredient 2: Mixed-Precision Induces Inference-Training Trade-Offs

The above scaling law suggests that, given a set of scaling parameters and a target loss we wish the model to achieve, we can directly solve for the "optimal" forward and backward precisions which allow us to match the loss. However, as pointed out by Sardana et al. [33], it is often the case in practice that we wish to put a larger weight on inference cost, rather than training cost, which can lead to different results when determining the "optimal" training precisions.

Operation	FP4:FP8	FP8:FP4	FP4:FP4
Forward / Inference (spfw)	2.0	1.0	2.0
Backward (spbw)	1.0	2.0	2.0
Training (sptr)	1.2	1.5	2.0

Table 1. Speedups relative to an FP8 baseline for forward (spfw), backward (spbw); sptr is the harmonic mean of spfw and spbw with weights $1/3$ (forward) and $2/3$ (backward).

Because inference latency depends solely on the *forward* pass ($\sim 33\%$ of training compute) while the *backward* pass consumes the remaining $\sim 66\%$, these trade-offs may need to be analyzed separately.

Specifically, we can state a set of simple guiding principles:

- **Forward pass.** Low-precision induces a trade-off between reduced *parameter efficiency*, and increased inference speed: for instance, we could train a larger model in terms of parameters N, but quantize its forward pass to lower precision, and obtain a better trade-off. As such, P_{fw} should be picked to optimize this trade-off.
- **Backward pass.** Similarly, *training speedup due to a quantized backward pass* can offset the *reduced data efficiency* eff_D: we could train more heavily-quantized model *on more data* under the same computing budget. Thus, P_{bw} should be picked to optimize this trade-off.

We contrast this with previous work, which often requires lower precision to suffer *no* accuracy loss (e.g., Chmiel et al. [11]). This unnecessarily reduces these trade-offs to simple selection of the fastest lossless precision. We argue that scaling-law analysis enables a more fine-grained approach needed to decide upon the "optimal" set of forward and backward precisions.

Example speedup model. To illustrate this, we assume a hardware-agnostic bit-wise ops (BOPS) model, which states that speedup is inversely proportional to datatype bit-width. The speedups are stated in Table 1, relative to FP8:

Given a forward-pass compute budget $N_{\max}$ and a training budget $N_{\max} D_{\max}$, the effective loss will be given by:

$$Loss\left(N_{\max}\ \text{spfw},\ D_{\max}\ \text{sptr}\,/\,\text{spfw},\ P_{\text{fw}},\ P_{\text{bw}}\right),$$

which we evaluate with the scaling law from Equation (1), leading to the fit from Figure1(a). One can see how spfw and sptr propagate as multiplicative factors on eff_N and eff_D and directly counter the suboptimal parameter and data efficiencies. Figures 1(b)–(c) illustrate the optimality regions: specifically, it tells us for which model sizes (Y axis) and corresponding data-to-model ratios (X axis) FP4 is optimal relative to FP8 (red vs orange region). The green thatched area is the range in which *training using our MXFP4 implementation* would be optimal by this metric.

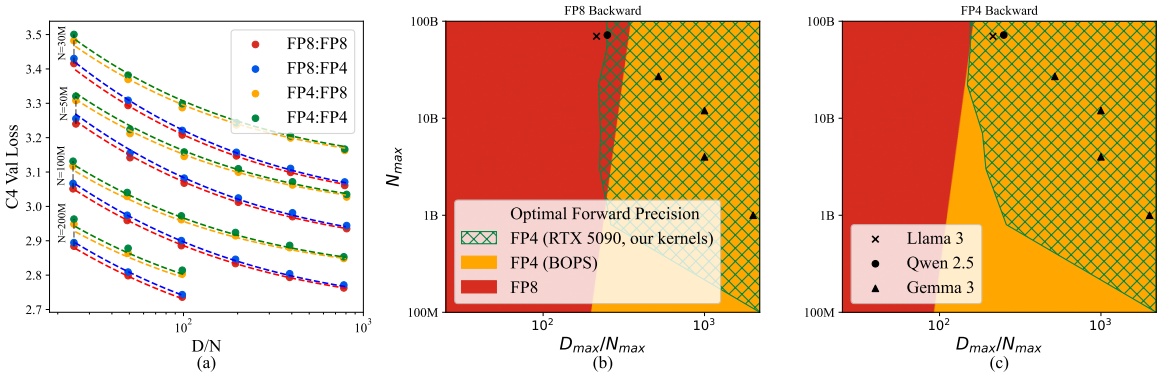

Figure 1. Analysis of Quartet: **(a)** Scaling-law 1 fit for various FORWARD:BACKWARD precisions. **(b)** Regions where each *forward* precision is optimal when the *backward* pass is FP8. **(c)** Same, but with an FP4 backward pass. Observe that the FP4 backward enlarges the regime in which FP4 forward is optimal. Interestingly, popular models such as Llama3 or Qwen2.5 fall into the FP4 optimality region, implying that training similar models in FP4 might have been optimal.

(This is derived using our actual obtained speedups.)

Ingredient 2 says that *low-precision impact should be analysed under the compute budget*; scaling-law fits then reveal when a given precision is the optimal choice for either pass.

3. Ingredient 3: Minimal Forward-Pass Error with Unbiased Gradient Estimation

The above ingredients should allow us to determine the "best" quantized training method among existing approaches, focusing on the hardware-supported MXFP4 [31] format.

Forward Pass Quantization. As detailed in Section A, existing QAT (forward-only) approaches can be split into "noise injection" [5] and "error-minimization" approaches, e.g. [32]. Focusing on the forward pass, by the above discussion (Ingredients 1 and 2), we seek the approach which maximizes the parameter efficiency factor eff_N. For this, we implement four standard schemes for QAT: 1) stochastic rounding (SR) with standard AbsMax per-group normalization [39]; 2) vanilla round-to-nearest (RTN) quantization with AbsMax per-group normalization; 3) learnable scale clipping (LSQ) with RTN quantization [16; 48]; 4) Hadamard normalization followed by RMSE-based clipping (QuEST) [32]. We apply the Hadamard transform to weights and activations for each one of these schemes before quantization. We compare these approaches following Section 1: we train models using each technique, apply scaling law fitting, and register their resulting eff_N factors. For additional information, we also show representations' mean-squared error (MSE) for fitting random Gaussian data. The results are provided in the first rows/columns of Table 2.

The results in Table 2 show that QuEST has the best parameter efficiency eff_N among all existing methods. Moreover, eff_N appears to correlate heavily with MSE, as suggested by [32]. Additionally, the results align with the analysis of Chmiel et al. [11] that determined deterministic RTN to always be preferable to stochastic rounding for the forward.

Rounding	eff_N	MSE	eff^*_D	Misalignment
Stochastic Rounding AbsMax	0.44	2.84×10^{-2}	**0.85**	0
Round-to-nearest AbsMax	0.61	1.40×10^{-2}	0.83	9.3×10^{-3}
QuEST (Hadamard + RMSE)	**0.65**	1.35×10^{-2}	0.18	1.3×10^{-2}
RTN AbsMax PMA	0.61	1.42×10^{-2}	0.83	2.8×10^{-5}

Table 2. Illustration of error-bias trade-off between different quantized forward and backward pass approaches. For the forward (given by the eff_N metric) the best performing method is QuEST, correlating with superior MSE over Gaussian input data. By contrast, for the backward pass (the data efficiency eff^*_D computed at 800 Tokens/Parameter), the best performing method is stochastic rounding, correlated with perfect magnitude alignment. This justifies our choice of method, which combines block-wise QuEST on the forward, with Stochastic Rounding on the backward pass.

Backward pass: a novel error-bias trade-off. The above findings do not transfer to backward pass quantization, as optimization theory shows that unbiased gradient estimation is critical for convergence, e.g. [1].

To study gradient bias, we follow the analysis of [40; 41], who studied RTN quantization with randomized rotations, approximated by the randomized Hadamard transform, which we denote by $\widehat{H}$. They show that, while RHT makes quantization unbiased *in direction*, it adds a bias *in magnitude*. To address this, they introduce fine group-wise scaling factors S that make the estimation truly unbiased.

$$\mathbb{E}_\xi[Q(X,\xi)] = X \text{ if } Q(X,\xi) = S(X) \cdot \text{RTN}(\widehat{H}(X,\xi)).$$

Unfortunately, their re-scaling is incompatible with coarse

group-wise scaling of the MXFP4 format, so we cannot use it in practice. However, we can still use their approach to gauge the degree of misalignment for different quantizers by simply studying their corresponding expected value of $\mathbb{E}[1/S]$, which we call the *projection magnitude alignment (PMA)*. Misalignment ($1 - \mathbb{E}[1/S]$) is shown in Table 2, along with the MSE across different schemes. Focusing on stochastic rounding vs round-to-nearest with AbsMax, one can see that SR trades high error for perfect alignment.

Additional experiments aimed at bridging the gap between PMA and final performance are presented in Appendix C. In short, we observe that MSE has high impact on initial convergence and shorter training runs, while PMA has greater impact on longer runs. Concretely, while RTN backward quantization may be preferable for shorter training, stochastic rounding (SR) performs consistently better for models more saturated with data. In this setup, the inflection point is around the $D/N = 400$ data-to-parameter ratio.

Summary. Our analysis outlines a new trade-off between parameter efficiency on the forward (equated with quantization MSE), and data-efficiency on the backward (which we equate with the new misalignment metric). In the following, we will adopt a "best of both worlds" approach, aiming to perform a forward pass that minimizes MSE (based on QuEST [32]) together with a backward pass that is unbiased (based on Stochastic Rounding [39]). The novel challenge, which we address next, will be an extremely efficient GPU-aware implementation of such an approach.

4. Ingredient 4: Fast GPU Support for Accurate Quantized Training

Algorithm 1 Quartet MXFP4 Forward-Backward Algorithm

Require: Hadamard Transform (H_g, $\widehat{\mathrm{H}}_g$) block size g

function FORWARD(input X, weights W)
  $X_h \leftarrow \mathrm{H}_g(X)$; $W_h \leftarrow \mathrm{H}_g(W)$
  $(X_q, \alpha_x) \leftarrow \mathrm{QuEST}(X_h)$
  $(W_q, \alpha_w) \leftarrow \mathrm{QuEST}(W_h)$
  $Y_q \leftarrow \mathrm{GEMM}_{\mathrm{LP}}(X_q, W_q)$
  $y \leftarrow (\alpha_x \alpha_w) \cdot \mathrm{RESCALE}(Y_q)$
  return y, ctx $= \{X_q, W_q, \alpha_x, \alpha_w\}$
end function
function BACKWARD(output gradient dy, ctx, random seed ξ)
  Unpack $\{X_q, W_q, \alpha_x, \alpha_w\}$ from ctx
  $G_h \leftarrow \widehat{\mathrm{H}}_g(dy, \xi)$; $W_h^\top \leftarrow \widehat{\mathrm{H}}_g(W_q^\top, \xi)$
  $G_q \leftarrow \mathrm{SR}(\frac{3}{4} G_h)$; $W_q^\top \leftarrow \mathrm{SR}(\frac{3}{4} W_h^\top)$
  $dx_q \leftarrow \mathrm{GEMM}_{\mathrm{LP}}(G_q, W_q^\top)$
  $dx \leftarrow \mathrm{H}_g^{-1}\big(\frac{16}{9} dx_q \odot \alpha_x\big)$
  $G_h^\top \leftarrow \widehat{\mathrm{H}}_g(dy^\top, \xi)$; $X_h^\top \leftarrow \widehat{\mathrm{H}}_g(X_q^\top, \xi)$
  $G_q^\top \leftarrow \mathrm{SR}(\frac{3}{4} G_h^\top)$; $X_q^\top \leftarrow \mathrm{SR}(\frac{3}{4} X_h^\top)$
  $dW_q \leftarrow \mathrm{GEMM}_{\mathrm{LP}}(G_q^\top, X_q^\top)$
  $dW \leftarrow \mathrm{H}_g^{-1}\big(\frac{16}{9} dW_q \odot \alpha_w\big)$
  return dx, dW
end function

Quartet Overview. We integrate our prior discussion into Algorithm 1, which aims to perform accurate training while executing *all three* matrix multiplications of a linear layer in low precision. The **forward pass** applies a fixed Hadamard transform H_g (of block size g equal to the quantization group size) and QuEST projection to low precision and multiplies the resulting tensors with an MXFP4 kernel. The **backward pass** decorrelates the multiplied tensors with an identical block-wise random Hadamard transform $\widehat{\mathrm{H}}_g$, applies unbiased stochastic rounding (SR) to MXFP4, performs the two gradient GEMMs in MXFP4, rescales to compensate for SR range matching, applies QuEST masking and inverts the Hadamard transform H_g.

Costs and Format Specialization. The key added cost of the above pipeline is that of the Hadamard rotations and their inversion: specifically, two Hadamard/Inverse transforms are added over standard training. Our key observation is that, since the MXFP4 already groups 32 consecutive weights (in 1D), sharing scales, we can and should apply the Hadamard rotations and their inversion at the same group size. With a fast Hadamard implementation, the theoretical cost is $O(g \log g)$—negligible for $g \leq 256$ compared with the GEMMs.

GPU Kernel Support. While the above blueprint appears simple, implementing it efficiently on Blackwell GPUs—in order to leverage fast MXFP4 support—is extremely challenging. For illustration, a direct implementation of the above pattern would be *slower* than FP16 unquantized training, let alone optimized FP8. Our fast implementation builds on CUTLASS 3.9 [37], which provides templates for the new Blackwell architecture. Computation happens in two stages: **Stage 1** fuses the Hadamard transform, quantization, scale calculation, and QuEST clipping mask generation (only on forward) into a single kernel; **Stage 2** performs GEMM using a dedicated kernel.

To our knowledge, our implementation is the first to efficiently support quantization-related operations on the Blackwell architecture.

5. Experiments

Experimental Setup and Scaling Law Fit. As described in Section B, we pre-train Llama-style models on C4 and report validation loss after a fixed token budget. All baselines reuse the optimiser, schedule, and hyper-parameters, as described in Appendix E. Following Section 1, we compare accuracy across methods by fitting the full scaling law in Eqn. 1 across methods, as follows: we fit parameters A, α, B, β, E and γ on a grid of baseline precision runs (FP8 forward, FP8 backward) shown on Figure 1(a). Then we fit the parameter and data efficiencies eff_N and eff_D separately for every forward and backward quantization scheme we evaluate. The law is fitted identically to prior work in

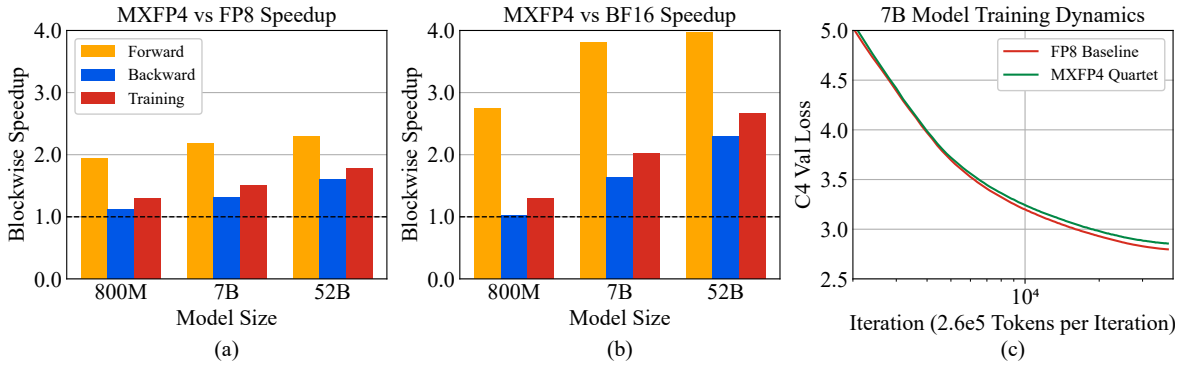

Figure 2. **(a, left), (b, middle)**: Quartet kernels block-wise speedup across model sizes relative to FP8 and BF16. **(c, right):** Training dynamics for the 7B model trained with Quartet relative to FP8 .

Method	25×	50×	100×	200×	400×	eff_N	eff_D
LUQ–INT4	3.729	3.684	3.658	3.432	3.399	0.50	0.15
LUQ–FP4	4.806	4.906	4.880	4.842	4.799	0.01	0.09
Jetfire–FP4	7.033	6.941	6.759	6.621	6.581	0.01	0.07
HALO–FP4	6.649	7.040	6.551	6.501	5.381	Unstable	
LSS–INT4	NaN	3.398	NaN	NaN	NaN	Unstable	
Quartet (ours)	**3.500**	**3.382**	**3.299**	**3.244**	**3.205**	**0.64**	**0.94**

Table 3. Validation loss (lower is better) on C4 for Llama models with 30M parameters and efficiency coefficients fitted on them. Columns show the tokens-to-parameters ratio (D/N). All methods share identical setups; only the quantization scheme varies. NaNs for LLS-INT4 appeared at arbitrary stages of training without any irregularities.

this area [23; 26; 8]. A detailed description is presented in Appendix F.

Accuracy Comparisons. We compare accuracy (validation loss) as well as the efficiency factors against four recent, fully–quantized training pipelines that operate in 4-bit precision for *both* forward and backward passes: 1) **LUQ** [11] applies to both INT4 and FP4, using unbiased quantization that pairs 4-bit weights/activations with stochastic underflow, and logarithmic stochastic rounding; 2) **HALO** [3], which uses Hadamard rotations to mitigate outliers, evaluated in FP4 at their most accurate HALO-2 setting; 3) **Jetfire** [50] quantizes in blocks of 32×32, originally introduced for INT8, and adapted to FP4 for our setup; 4) **LSS** [48] for INT4 training, that combines a Hadamard-based forward pass with "leverage–score" sampled INT4 gradients.

Accuracy Discussion. As can be seen in Table 3, across all token-to-parameter ratios, Quartet attains the lowest loss, often by very large margins. At 100× toks/param., Quartet improves upon LUQ–INT4 by 10% relative loss, and the gap widens as we increase data size. We note that Jetfire and HALO incur large degradation and are unstable when ported to FP4. Interestingly, LSS is competitive only for shorter runs, and diverges for longer training budgets, beyond 50×, matching observations from prior work [18]. Overall, LUQ–INT4 is the strongest prior work; however, Quartet reaches significantly higher parameter and data efficiency, suggesting that it requires, roughly, 15% fewer parameters and 5x less data to reach the same loss. Figure 2 (c) additionally demonstrates the stability of Quartet for training models two orders of magnitude larger (7B).

Speedup Results. Next, we evaluate the efficiency of our implementation on the NVIDIA RTX 5090 GPU by measuring its performance across single layers of standard shapes, and aggregating across an entire Transformer block. Speedup results are shown in Figure 2, using a batch size 64 and sequence length of 512. The FP8 baseline is provided by CUTLASS MXFP8 kernels, while the BF16 baseline uses PyTorch, both using Blackwell-optimized kernels. Inference speedups are more pronounced due to the lower cost of the forward pass compared to the backward pass, and the latter's higher computational complexity. The speedup scales with the arithmetic intensity (i.e., model size), reaching up to 2.4× over FP8 and 4× over BF16 on the forward pass, where it stabilizes. In the backward pass, our implementation achieves up to 1.6× over FP8 and 2.3× over BF16, resulting in an overall training speedup of up to around 1.8×, and 2.6×, respectively.

6. Discussion and Limitations

We provided a set of guidelines to modeling, comparing and designing fully-quantized training schemes for large language models. Moreover, we followed those guidelines to arrive at Quartet: a new SOTA full MXFP4 training algorithm. One current limiting factor is that Quartet was designed with a specific (standard) data-type and compute architecture in mind. Certain aspects of our method rely on specialized operations, like stochastic rounding, which have hardware support for MXFP4, but may be lacking for other formats. In future work, we plan to look into generalizing our approach to alternative formats, as well as larger-scale

distributed model execution.

References

[1] Alistarh, D., Grubic, D., Li, J., Tomioka, R., and Vojnovic, M. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in neural information processing systems*, 30, 2017.

[2] Ashkboos, S., Mohtashami, A., Croci, M. L., Li, B., Cameron, P., Jaggi, M., Alistarh, D., Hoefler, T., and Hensman, J. Quarot: Outlier-free 4-bit inference in rotated llms. *arXiv preprint arXiv:2404.00456*, 2024. URL https://arxiv.org/abs/2404.00456.

[3] Ashkboos, S., Nikdan, M., Tabesh, S., Castro, R. L., Hoefler, T., and Alistarh, D. Halo: Hadamard-assisted lower-precision optimization for llms, 2025. URL https://arxiv.org/abs/2501.02625.

[4] Banner, R., Hubara, I., Hoffer, E., and Soudry, D. Scalable methods for 8-bit training of neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[5] Baskin, C., Liss, N., Schwartz, E., Zheltonozhskii, E., Giryes, R., Bronstein, A. M., and Mendelson, A. Uniq: Uniform noise injection for non-uniform quantization of neural networks. *ACM Transactions on Computer Systems (TOCS)*, 37(1-4):1–15, 2021.

[6] Bengio, Y., Léonard, N., and Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[7] Bhalgat, Y., Lee, J., Nagel, M., Blankevoort, T., and Kwak, N. Lsq+: Improving low-bit quantization through learnable offsets and better initialization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

[8] Busbridge, D., Shidani, A., Weers, F., Ramapuram, J., Littwin, E., and Webb, R. Distillation scaling laws, 2025. URL https://arxiv.org/abs/2502.08606.

[9] Chee, J., Cai, Y., Kuleshov, V., and Sa, C. D. Quip: 2-bit quantization of large language models with guarantees. *arXiv preprint arXiv:2307.13304*, 2023. URL https://arxiv.org/abs/2307.13304.

[10] Chmiel, B., Banner, R., Hoffer, E., Ben-Yaacov, H., and Soudry, D. Accurate Neural Training with 4-bit Matrix Multiplications at Standard Formats. In *International Conference on Learning Representations (ICLR)*, 2023.

[11] Chmiel, B., Banner, R., Hoffer, E., Yaacov, H. B., and Soudry, D. Accurate neural training with 4-bit matrix multiplications at standard formats, 2024. URL https://arxiv.org/abs/2112.10769.

[12] Choi, J., Wang, Z., Venkataramani, S., Chuang, P. I.-J., Srinivasan, V., and Gopalakrishnan, K. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

[13] Dao-AILab. Fast hadamard transform in cuda, with a pytorch interface. https://github.com/Dao-AILab/fast-hadamard-transform, 2024. Accessed: 2025-05-13.

[14] Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. LLM.int8(): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022. URL https://arxiv.org/abs/2208.07339.

[15] Dodge, J., Sap, M., Marasović, A., Agnew, W., Ilharco, G., Groeneveld, D., Mitchell, M., and Gardner, M. Documenting large webtext corpora: A case study on the colossal clean crawled corpus, 2021. URL https://arxiv.org/abs/2104.08758.

[16] Esser, S. K., McKinstry, J. L., Bablani, D., Appuswamy, R., and Modha, D. S. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.

[17] Fino and Algazi. Unified matrix treatment of the fast walsh-hadamard transform. *IEEE Transactions on Computers*, 100(11):1142–1146, 1976.

[18] Fishman, M., Chmiel, B., Banner, R., and Soudry, D. Scaling fp8 training to trillion-token llms. *arXiv preprint arXiv:2409.12517*, 2024.

[19] Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. GPTQ: Accurate post-training compression for generative pretrained transformers. *arXiv preprint arXiv:2210.17323*, 2022. URL https://arxiv.org/abs/2210.17323.

[20] Frantar, E., Ruiz, C. R., Houlsby, N., Alistarh, D., and Evci, U. Scaling laws for sparsely-connected foundation models. In *International Conference on Learning Representations*, 2024.

[21] Frantar, E., Evci, U., Park, W., Houlsby, N., and Alistarh, D. Compression scaling laws:unifying sparsity and quantization, 2025. URL https://arxiv.org/abs/2502.16440.

[22] Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., Yang, A., Fan, A., Goyal, A., Hartshorn, A., Yang, A., Mitra, A., Sravankumar, A.,

Korenev, A., Hinsvark, A., Rao, A., Zhang, A., Rodriguez, A., Gregerson, A., Spataru, A., Roziere, B., Biron, B., Tang, B., Chern, B., Caucheteux, C., Nayak, C., Bi, C., Marra, C., McConnell, C., Keller, C., Touret, C., Wu, C., Wong, C., Ferrer, C. C., Nikolaidis, C., Allonsius, D., Song, D., Pintz, D., Livshits, D., Wyatt, D., Esiobu, D., Choudhary, D., Mahajan, D., Garcia-Olano, D., Perino, D., Hupkes, D., Lakomkin, E., AlBadawy, E., Lobanova, E., Dinan, E., Smith, E. M., Radenovic, F., Guzmán, F., Zhang, F., Synnaeve, G., Lee, G., Anderson, G. L., Thattai, G., Nail, G., Mialon, G., Pang, G., Cucurell, G., Nguyen, H., Korevaar, H., Xu, H., Touvron, H., Zarov, I., Ibarra, I. A., Kloumann, I., Misra, I., Evtimov, I., Zhang, J., Copet, J., Lee, J., Geffert, J., Vranes, J., Park, J., Mahadeokar, J., Shah, J., van der Linde, J., Billock, J., Hong, J., Lee, J., Fu, J., Chi, J., Huang, J., Liu, J., Wang, J., Yu, J., Bitton, J., Spisak, J., Park, J., Rocca, J., Johnstun, J., Saxe, J., Jia, J., Alwala, K. V., Prasad, K., Upasani, K., Plawiak, K., Li, K., Heafield, K., Stone, K., El-Arini, K., Iyer, K., Malik, K., Chiu, K., Bhalla, K., Lakhotia, K., Rantala-Yeary, L., van der Maaten, L., Chen, L., Tan, L., Jenkins, L., Martin, L., Madaan, L., Malo, L., Blecher, L., Landzaat, L., de Oliveira, L., Muzzi, M., Pasupuleti, M., Singh, M., Paluri, M., Kardas, M., Tsimpoukelli, M., Oldham, M., Rita, M., Pavlova, M., Kambadur, M., Lewis, M., Si, M., Singh, M. K., Hassan, M., Goyal, N., Torabi, N., Bashlykov, N., Bogoychev, N., Chatterji, N., Zhang, N., Duchenne, O., Çelebi, O., Alrassy, P., Zhang, P., Li, P., Vasic, P., Weng, P., Bhargava, P., Dubal, P., Krishnan, P., Koura, P. S., Xu, P., He, Q., Dong, Q., Srinivasan, R., Ganapathy, R., Calderer, R., Cabral, R. S., Stojnic, R., Raileanu, R., Maheswari, R., Girdhar, R., Patel, R., Sauvestre, R., Polidoro, R., Sumbaly, R., Taylor, R., Silva, R., Hou, R., Wang, R., Hosseini, S., Chennabasappa, S., Singh, S., Bell, S., Kim, S. S., Edunov, S., Nie, S., Narang, S., Raparthy, S., Shen, S., Wan, S., Bhosale, S., Zhang, S., Vandenhende, S., Batra, S., Whitman, S., Sootla, S., Collot, S., Gururangan, S., Borodinsky, S., Herman, T., Fowler, T., Sheasha, T., Georgiou, T., Scialom, T., Speckbacher, T., Mihaylov, T., Xiao, T., Karn, U., Goswami, V., Gupta, V., Ramanathan, V., Kerkez, V., Gonguet, V., Do, V., Vogeti, V., Albiero, V., Petrovic, V., Chu, W., Xiong, W., Fu, W., Meers, W., Martinet, X., Wang, X., Wang, X., Tan, X. E., Xia, X., Xie, X., Jia, X., Wang, X., Goldschlag, Y., Gaur, Y., Babaei, Y., Wen, Y., Song, Y., Zhang, Y., Li, Y., Mao, Y., Coudert, Z. D., Yan, Z., Chen, Z., Papakipos, Z., Singh, A., Srivastava, A., Jain, A., Kelsey, A., Shajnfeld, A., Gangidi, A., Victoria, A., Goldstand, A., Menon, A., Sharma, A., Boesenberg, A., Baevski, A., Feinstein, A., Kallet, A., Sangani, A., Teo, A., Yunus, A., Lupu, A., Alvarado, A., Caples, A., Gu, A., Ho, A., Poulton, A., Ryan, A., Ramchandani, A., Dong, A., Franco, A., Goyal, A., Saraf, A., Chowdhury, A., Gabriel, A., Bharambe, A., Eisenman, A., Yazdan, A., James, B., Maurer, B., Leonhardi, B., Huang, B., Loyd, B., Paola, B. D., Paranjape, B., Liu, B., Wu, B., Ni, B., Hancock, B., Wasti, B., Spence, B., Stojkovic, B., Gamido, B., Montalvo, B., Parker, C., Burton, C., Mejia, C., Liu, C., Wang, C., Kim, C., Zhou, C., Hu, C., Chu, C.-H., Cai, C., Tindal, C., Feichtenhofer, C., Gao, C., Civin, D., Beaty, D., Kreymer, D., Li, D., Adkins, D., Xu, D., Testuggine, D., David, D., Parikh, D., Liskovich, D., Foss, D., Wang, D., Le, D., Holland, D., Dowling, E., Jamil, E., Montgomery, E., Presani, E., Hahn, E., Wood, E., Le, E.-T., Brinkman, E., Arcaute, E., Dunbar, E., Smothers, E., Sun, F., Kreuk, F., Tian, F., Kokkinos, F., Ozgenel, F., Caggioni, F., Kanayet, F., Seide, F., Florez, G. M., Schwarz, G., Badeer, G., Swee, G., Halpern, G., Herman, G., Sizov, G., Guangyi, Zhang, Lakshminarayanan, G., Inan, H., Shojanazeri, H., Zou, H., Wang, H., Zha, H., Habeeb, H., Rudolph, H., Suk, H., Aspegren, H., Goldman, H., Zhan, H., Damlaj, I., Molybog, I., Tufanov, I., Leontiadis, I., Veliche, I.-E., Gat, I., Weissman, J., Geboski, J., Kohli, J., Lam, J., Asher, J., Gaya, J.-B., Marcus, J., Tang, J., Chan, J., Zhen, J., Reizenstein, J., Teboul, J., Zhong, J., Jin, J., Yang, J., Cummings, J., Carvill, J., Shepard, J., McPhie, J., Torres, J., Ginsburg, J., Wang, J., Wu, K., U, K. H., Saxena, K., Khandelwal, K., Zand, K., Matosich, K., Veeraraghavan, K., Michelena, K., Li, K., Jagadeesh, K., Huang, K., Chawla, K., Huang, K., Chen, L., Garg, L., A, L., Silva, L., Bell, L., Zhang, L., Guo, L., Yu, L., Moshkovich, L., Wehrstedt, L., Khabsa, M., Avalani, M., Bhatt, M., Mankus, M., Hasson, M., Lennie, M., Reso, M., Groshev, M., Naumov, M., Lathi, M., Keneally, M., Liu, M., Seltzer, M. L., Valko, M., Restrepo, M., Patel, M., Vyatskov, M., Samvelyan, M., Clark, M., Macey, M., Wang, M., Hermoso, M. J., Metanat, M., Rastegari, M., Bansal, M., Santhanam, N., Parks, N., White, N., Bawa, N., Singhal, N., Egebo, N., Usunier, N., Mehta, N., Laptev, N. P., Dong, N., Cheng, N., Chernoguz, O., Hart, O., Salpekar, O., Kalinli, O., Kent, P., Parekh, P., Saab, P., Balaji, P., Rittner, P., Bontrager, P., Roux, P., Dollar, P., Zvyagina, P., Ratanchandani, P., Yuvraj, P., Liang, Q., Alao, R., Rodriguez, R., Ayub, R., Murthy, R., Nayani, R., Mitra, R., Parthasarathy, R., Li, R., Hogan, R., Battey, R., Wang, R., Howes, R., Rinott, R., Mehta, S., Siby, S., Bondu, S. J., Datta, S., Chugh, S., Hunt, S., Dhillon, S., Sidorov, S., Pan, S., Mahajan, S., Verma, S., Yamamoto, S., Ramaswamy, S., Lindsay, S., Lindsay, S., Feng, S., Lin, S., Zha, S. C., Patil, S., Shankar, S., Zhang, S., Zhang, S., Wang, S., Agarwal, S., Sajuyigbe, S., Chintala, S., Max, S., Chen, S., Kehoe, S., Satterfield, S., Govindaprasad, S., Gupta, S., Deng, S., Cho, S., Virk, S., Subramanian, S., Choudhury, S., Goldman, S., Remez, T., Glaser,

T., Best, T., Koehler, T., Robinson, T., Li, T., Zhang, T., Matthews, T., Chou, T., Shaked, T., Vontimitta, V., Ajayi, V., Montanez, V., Mohan, V., Kumar, V. S., Mangla, V., Ionescu, V., Poenaru, V., Mihailescu, V. T., Ivanov, V., Li, W., Wang, W., Jiang, W., Bouaziz, W., Constable, W., Tang, X., Wu, X., Wang, X., Wu, X., Gao, X., Kleinman, Y., Chen, Y., Hu, Y., Jia, Y., Qi, Y., Li, Y., Zhang, Y., Zhang, Y., Adi, Y., Nam, Y., Yu, Wang, Zhao, Y., Hao, Y., Qian, Y., Li, Y., He, Y., Rait, Z., DeVito, Z., Rosnbrick, Z., Wen, Z., Yang, Z., Zhao, Z., and Ma, Z. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

[23] Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., and Sifre, L. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

[24] Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

[25] Kaushal, A., Vaidhya, T., Mondal, A. K., Pandey, T., Bhagat, A., and Rish, I. Spectra: Surprising effectiveness of pretraining ternary language models at scale. *arXiv preprint arXiv:2407.12327*, 2024.

[26] Kumar, T., Ankner, Z., Spector, B. F., Bordelon, B., Muennighoff, N., Paul, M., Pehlevan, C., Ré, C., and Raghunathan, A. Scaling laws for precision, 2024. URL https://arxiv.org/abs/2411.04330.

[27] Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Guo, D., Yang, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Zhang, H., Ding, H., Xin, H., Gao, H., Li, H., Qu, H., Cai, J. L., Liang, J., Guo, J., Ni, J., Li, J., Wang, J., Chen, J., Chen, J., Yuan, J., Qiu, J., Li, J., Song, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Xu, L., Xia, L., Zhao, L., Wang, L., Zhang, L., Li, M., Wang, M., Zhang, M., Zhang, M., Tang, M., Li, M., Tian, N., Huang, P., Wang, P., Zhang, P., Wang, Q., Zhu, Q., Chen, Q., Du, Q., Chen, R. J., Jin, R. L., Ge, R., Zhang, R., Pan, R., Wang, R., Xu, R., Zhang, R., Chen, R., Li, S. S., Lu, S., Zhou, S., Chen, S., Wu, S., Ye, S., Ma, S., Wang, S., Zhou, S., Yu, S., Zhou, S., Pan, S., Wang, T., Yun, T., Pei, T., Sun, T., Xiao, W. L., Zeng, W., Zhao, W., An, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Li, X. Q., Jin, X., Wang, X., Bi, X., Liu, X., Wang, X., Shen, X., Chen, X., Zhang, X., Chen, X., Nie, X., Sun, X., Wang, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yu, X., Song, X., Shan, X., Zhou, X., Yang, X., Li, X., Su, X., Lin, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhu, Y. X., Zhang, Y., Xu, Y., Huang, Y., Li, Y., Zhao, Y., Sun, Y., Li, Y., Wang, Y., Yu, Y., Zheng, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Tang, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Wu, Y., Ou, Y., Zhu, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Zha, Y., Xiong, Y., Ma, Y., Yan, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Wu, Z. F., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Huang, Z., Zhang, Z., Xie, Z., Zhang, Z., Hao, Z., Gou, Z., Ma, Z., Yan, Z., Shao, Z., Xu, Z., Wu, Z., Zhang, Z., Li, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Gao, Z., and Pan, Z. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024. URL https://arxiv.org/abs/2412.19437.

[28] Loshchilov, I. and Hutter, F. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

[29] Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., and Wu, H. Mixed precision training, 2018. URL https://arxiv.org/abs/1710.03740.

[30] NVIDIA Corporation. Nvidia blackwell architecture technical brief. https://resources.nvidia.com/en-us-blackwell-architecture, 2024. Accessed: 2025-05-13.

[31] Open Compute Project. Ocp microscaling formats (mx) specification version 1.0. https://www.opencompute.org/documents/ocp-microscaling-formats-mx-v1-0-spec-final-pdf 2023. Accessed: 2025-05-13.

[32] Panferov, A., Chen, J., Tabesh, S., Castro, R. L., Nikdan, M., and Alistarh, D. Quest: Stable training of llms with 1-bit weights and activations, 2025. URL https://arxiv.org/abs/2502.05003.

[33] Sardana, N., Portes, J., Doubov, S., and Frankle, J. Beyond chinchilla-optimal: Accounting for inference in language model scaling laws, 2025. URL https://arxiv.org/abs/2401.00448.

[34] Sun, X., Wang, N., Chen, C.-Y., Ni, J., Agrawal, A., Cui, X., Venkataramani, S., El Maghraoui, K., Srinivasan, V., and Gopalakrishnan, K. Ultra-Low Precision 4-bit Training of Deep Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[35] Suresh, A. T., Felix, X. Y., Kumar, S., and McMahan, H. B. Distributed mean estimation with limited communication. In *International conference on machine learning*, pp. 3329–3337. PMLR, 2017.

[36] Team, P. Hadacore: Accelerating large language models with fast hadamard transforms. https://pytorch.org/blog/hadacore/, 2024. Accessed: 2025-05-13.

[37] Thakkar, V., Ramani, P., Cecka, C., Shivam, A., Lu, H., Yan, E., Kosaian, J., Hoemmen, M., Wu, H., Kerr, A., Nicely, M., Merrill, D., Blasig, D., Qiao, F., Majcher, P., Springer, P., Hohnerbach, M., Wang, J., and Gupta, M. CUTLASS, January 2025. URL https://github.com/NVIDIA/cutlass.

[38] 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. URL https://arxiv.org/abs/2307.09288.

[39] Tseng, A., Yu, T., and Park, Y. Training llms with mxfp4, 2025. URL https://arxiv.org/abs/2502.20586.

[40] Vargaftik, S., Basat, R. B., Portnoy, A., Mendelson, G., Ben-Itzhak, Y., and Mitzenmacher, M. Drive: One-bit distributed mean estimation, 2021. URL https://arxiv.org/abs/2105.08339.

[41] Vargaftik, S., Basat, R. B., Portnoy, A., Mendelson, G., Ben-Itzhak, Y., and Mitzenmacher, M. Eden: Communication-efficient and robust distributed mean estimation for federated learning, 2022. URL https://arxiv.org/abs/2108.08842.

[42] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[43] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

[44] Wang, H., Ma, S., Dong, L., Huang, S., Wang, H., Ma, L., Yang, F., Wang, R., Wu, Y., and Wei, F. Bitnet: Scaling 1-bit transformers for large language models. *arXiv preprint arXiv:2310.11453*, 2023.

[45] Wang, R., Gong, Y., Liu, X., Zhao, G., Yang, Z., Guo, B., Zha, Z., and Cheng, P. Optimizing Large Language Model Training Using FP4 Quantization. *arXiv preprint arXiv:2501.17116*, 2024.

[46] Wortsman, M., Dettmers, T., Zettlemoyer, L., Morcos, A., Farhadi, A., and Schmidt, L. Stable and low-precision training for large-scale vision-language models. *Advances in Neural Information Processing Systems*, 36:10271–10298, 2023.

[47] Wortsman, M., Dettmers, T., Zettlemoyer, L., Morcos, A. S., Farhadi, A., and Schmidt, L. Stable and Low-Precision Training for Large-Scale Vision-Language Models. *arXiv preprint arXiv:2304.13013*, 2023.

[48] Xi, H., Li, C., Chen, J., and Zhu, J. Training Transformers with 4-bit Integers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[49] Xi, H., Chen, Y., Zhao, K., Zheng, K., Chen, J., and Zhu, J. Jetfire: Efficient and accurate transformer pretraining with int8 data flow and per-block quantization. *arXiv preprint arXiv:2403.12422*, 2024.

[50] Xi, H., Chen, Y., Zhao, K., Zheng, K., Chen, J., and Zhu, J. Jetfire: Efficient and Accurate Transformer Pretraining with INT8 Data Flow and Per-Block Quantization. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

[51] Yang, Y., Wu, S., Deng, L., Yan, T., Xie, Y., and Li, G. Training High-Performance and Large-Scale Deep Neural Networks with Full 8-bit Integers. *arXiv preprint arXiv:1909.02384*, 2020.

A. Related Work

Training in 8-bit formats. Early work on low-precision neural network training focused on 8-bit or higher precisions, mainly on CNNs. Banner et al. [4] demonstrated accurate 8-bit training via careful scaling and higher-precision accumulation. Yang et al. [51] proposed a framework that quantized weights, activations, gradients, errors, and even optimizer states to INT, achieving for the first time completely integer-only training with comparable accuracy. SwitchBack [47] and JetFire [50] build on this progress, targeting 8-bit training for Transformers [42]. Specifically, SwitchBack uses a hybrid INT8/BF16 linear layer for vision-language models, performing forward and input-gradient MatMuls in INT8 while computing weight gradients in 16-bit; this yielded 13–25% end-to-end speedups on CLIP models with accuracy within 0.1% of full precision.

JetFire [50] achieved *fully* INT8 training for Transformers by using a novel per-block quantization scheme to handle activation and gradient outliers. By partitioning matrices into small blocks and scaling each block independently, JetFire preserved accuracy comparable to FP16 training while obtaining $\sim 40\%$ end-to-end speedup and $1.49\times$ reduction in memory usage. The JetFire approach is conceptually similar to the FP8 DeepSeek training technique [27], which used larger block sizes. Recently, HALO [3] improved upon JetFire in terms of the accuracy-speedup trade-off in INT8, specifically focusing on low-precision fine-tuning. **In our work, we will treat FP8 as the lossless baseline for the purposes of comparison.**

End-to-end lower-precision training. As our results and prior work suggest, going below 8-bit precision in training using the above approaches is extremely challenging, due to narrower dynamic range and higher error. This frontier was first explored by Sun et al. [34], who achieved 4-bit training on ResNets by using a custom numeric format, which unfortunately is far from being supported in hardware. Chmiel et al. [10] introduced a logarithmic unbiased quantization (LUQ) scheme to this end, combining two prior ideas: (1) a log-scale FP4-type format to cover a wider dynamic range, and (2) applying stochastic unbiased rounding on the backward. For reference, LUQ incurs a 1.1% top-1 accuracy drop on ResNet50/ImageNet, and has not been validated on hardware-supported FP formats. Xi et al. [48] proposed a method to train Transformers using INT4 effective precision for all linear layers, using specialized quantizers: block-wise Hadamard transform and LSQ [16] for outlier mitigation on the forward pass, and leverage score sampling on the backward pass to exploit structured sparsity, together with a custom INT4-effective format. Their approach trains BERT-family models within 1-2% accuracy gap relative to FP16, with a 2.2x speedup on individual matrix multiplies (relative to 4x theoretical speedup), leading to up to 35% faster training end-to-end.

We compare relative to these techniques in Section 5, and show that Quartet outperforms them significantly in terms of accuracy and stability.

Mixed-precision training in low-precision formats. Given the importance of inference cost reductions, there has been significant work on *quantization-aware training (QAT)* [12; 7; 16; 5; 44; 25], i.e. methods that only quantize the *forward pass*. Two key difficulties in this setting are 1) minimizing the error induced by quantization on the forward pass, and 2) obtaining a stable gradient estimator over the resulting discrete space. With regards to error reduction, existing methods either try to find a good "learnable" fit w.r.t. the underlying continuous distribution [12; 16], or perform noise injection during QAT in order to make the network more robust to quantization [5]. Closer to our work, Wang et al. [45] explored FP4 QAT, introducing a "smoother" gradient estimator, together with outlier clamping and compensation to handle activation outliers. While their approach shows good accuracy, it is fairly complex and not validated in terms of efficient support. Concurrent work by [32] provided a simpler alternative approach, based on more precise MSE fitting, an optional Hadamard rotation, and a clipping-aware "trust" gradient estimator. By contrast with these forward-only approaches, recent work by Tseng et al. [39] investigated *backward-only* quantization with the MXFP4 format, signaling the importance of stochastic rounding and outlier mitigation in low-precision backpropagation.

B. Background

Quantization grids. Quantization maps high-precision internal model states, such as weights, activations, or gradients, to a lower-precision discrete set, i.e. the *quantization grid*. This grid can be *uniform*, e.g., for integer quantization, or *non-uniform*, e.g., floating-point (FP) quantization, where the value spacing is roughly exponential for fixed exponent. Since the original values may differ in scale compared to the grid, a higher-precision *scale* s is typically stored alongside the quantized values. For a vector x, the quantization process can be written as $q(x) = \text{round}\left(\frac{x}{s}; \text{grid}\right)$, and the original values can be approximately reconstructed as $\hat{x} = s \cdot q(x)$. Common choices for the scale are setting it to the maximum absolute value (absmax) in x (to avoid clipping) or optimizing it to minimize the mean squared quantization error, e.g. [32].

Quantization granularity. Apart from grid choice, quantization methods also differ in the *granularity* of the scales. A

single scale value can be shared across an entire tensor, e.g. [3], across each row or column [32], or over more fine-grained custom-defined blocks, such as 2D blocks [49; 27] or 1D blocks [31; 39]. Notably, the latest Blackwell GPU architecture [30] introduces hardware support for MXFP4/6/8 and NVFP4 formats. MXFP [31] formats share an FP8 power-of-two scale over each 1D block of 32 elements, while NVFP4 [30] uses FP8 (E4M3) scales and 1D blocks of 16 elements.

Rounding. Quantization typically involves rounding, e.g. via *deterministic rounding* to the nearest grid point, results in the lowest mean squared error (MSE). In contrast, *stochastic rounding* introduces randomness, rounding up or down with probabilities based on the input's distance to nearby grid points. While it may introduce higher MSE, stochastic rounding helps control bias, which can be crucial for maintaining the convergence of iterative optimization algorithms [1].

Outlier mitigation. One key issue when quantizing neural networks is the existence of large *outlier* values in the network weights, activations, and gradients [14]. One standard way of mitigating such outliers [35; 9; 3; 2; 39] is via the Hadamard transform: given a vector $x \in \mathbb{R}^d$, $h(x)$ is defined as $h(x) = H_d x$, where $H_d \in \mathbb{R}^{d\times d}$ is the normalized Hadamard matrix with elements from $\{\pm 1\}$. Hadamard matrices have a recursive structure $H_d = \frac{1}{\sqrt{2}} H_2 \otimes H_{d/2}$, which enables efficient computation when d is a power of two [17]. Optimized FWHT implementations for GPUs are available [13; 36]. When d is not a power of two, the input vector x is typically either zero-padded to the next power of two or transformed using a *Grouped Hadamard Transform*, where x is split into equal-sized blocks (each with power-of-two length), and the Hadamard transform is applied independently to each block.

Blackwell Architecture Support. NVIDIA's 5th-gen. Tensor Cores in Blackwell [30] provide native 4-bit floating-point execution. The cores support different block-scaled formats such as MXFP4 [31] and NVFP4, which roughly double the peak throughput over FP8/FP6, with a single B200 GPU peaking at 18 PFLOPS of dense FP4 compute [30]. Interestingly, our investigation shows that, as of now, MXFP4 is the only microscaling format with support for all required layouts for both forward and backward multiplications in low precision on Blackwell [37]. Therefore, we adopt MXFP4 for our implementation. This format stores each value using 1 sign bit + 1 mantissa bit + 2-bits for exponent. Every group of 32 elements shares a common 8-bit scaling factor, represented with 8 exponent bits, and no bits for mantissa. Blackwell's 5th-gen. Tensor Cores handle the required on-the-fly rescaling in hardware, without the need for software-based rescaling at CUDA level. Additional details are provided in Section 4.

LLM pre-training. We pre-train Transformers [43] of the Llama-2 [38] architecture in the range of 30, 50, 100, 200 million non-embedding parameters across a wide range of data-to-parameter ratios raging from 25x (around compute-optimal [23]) to 800x (extreme data saturation). We additionally selectively scale the model size up to around 7 billion parameters to verify training stability. We train all models on the train split of the C4 [15] dataset and report C4 validation loss as the main metric. We use the AdamW optimizer [28] weight decay of 0.1, gradient clipping of 1.0, a 10% LR warmup and cosine schedule. We identify the optimal LR for one of the small unquantized baseline models, scale it inverse-proportionally to the number of non-embedding parameters and reuse for every quantization scheme we evaluate. We present all hyper-parameters in Appendix E.

C. PMA Analysis

To connect PMA with training dynamics, we analyze the cumulative effect of misalignment and error on backward quantization for a 30M-parameters Llama model. In Figure 3 **(a)** and **(c)**, we plot the alignment metrics–Cosine Similarity and PMA—for inter-layer activation gradients as a function of back-propagation "depth". We can again observe the trade-off between similarity and magnitude alignment. Finally, Figure 3 **(c)** connects those quantities to final model quality (loss gap vs. full-precision model) for increasing data-vs-parameters.

D. Extra Kernel Information

Below, we provide additional description of the two-stage Quartet kernels.

Stage 1: Fused Quantization-Related Operations. First, we observe that, thanks to the small group size, the Hadamard transform can be implemented as a direct GEMM between the corresponding input matrix and a fixed 32×32 Hadamard matrix (see Sec. B), producing output in FP32, which is stored in GPU Shared Memory (SMEM). This allows us to implement the Hadamard operation efficiently by leveraging CUTLASS's multilevel tiling templates to optimize data movement. All subsequent operations are integrated via a custom CUTLASS *epilogue*, which utilizes the intermediate results previously stored in higher levels of the memory hierarchy and operates locally in the Register File (RF). At this stage, Blackwell's new hardware support is used to downcast FP32 values to FP4 (E2M1) using the PTX instructions for this

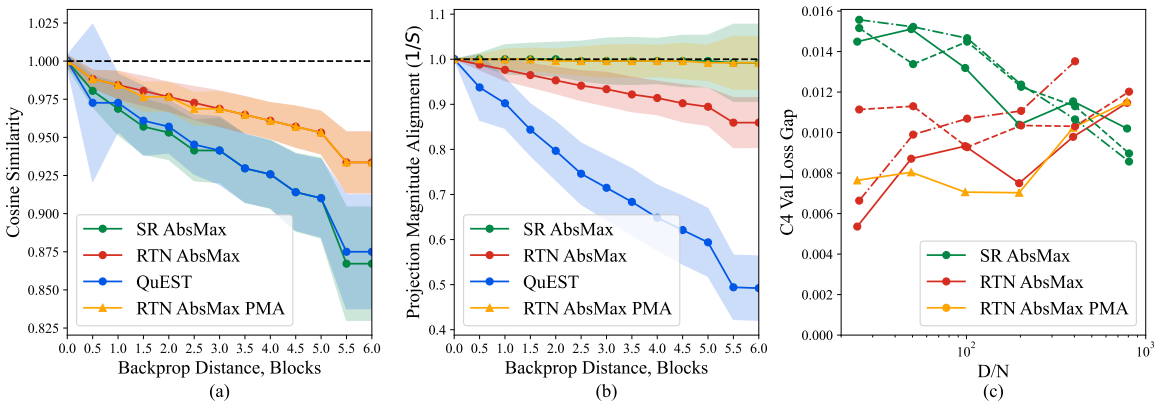

Figure 3. The effect of backward pass quantization on LLM training gradient quality and impact on performance: **(a, left)** and **(b, middle)** shows cosine similarity and projection magnitude alignement with unquantized reference, while **(c, right)** shows performance gaps with a non-quantized baseline for a set model sizes and data-to-parameter ratios (D/N).

purpose. To construct the final MXFP4 format, we compute scaling factors of shape 1×32. These scales are represented in 8-bit using the E8M0 format. Finally, the clipping mask is computed, and the three resulting tensors (values, scales, and mask) are written to Global Memory (GMEM). Throughout, data storage is optimized to use the widest memory instructions possible.

Stage 2: Dedicated GEMM Kernel. Blackwell introduces the `tcgen05.mma` instructions, which natively support matrix multiplication with scale factors in the form $D = C + (A \times \text{SFA}) \cdot (B \times \text{SFB})$. These scale factors are applied along the inner (K) dimension of the GEMM. For MXFP types, every 32 elements along the K-dimension of matrices A and B share a corresponding scale factor. This implies that an $M \times K$ matrix A is associated with a scale matrix SFA of size $M \times \lceil K/32 \rceil$. Our dedicated kernel is based on CUTLASS block-scaled GEMM for narrow precision. As part of this implementation, we also included the necessary functions to reorganize the scale factors generated in the Stage 1, aligning them with the layout required by this architecture [30].

E. 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_{layer})	6	7	8	10	32
Embedding Dimension (N_{embd})	640	768	1024	1280	4096
Attention Heads (N_{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 (γ)	0.1
Number of GPUs	8
Data Type (optimizer/accumulators)	`FP32`

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

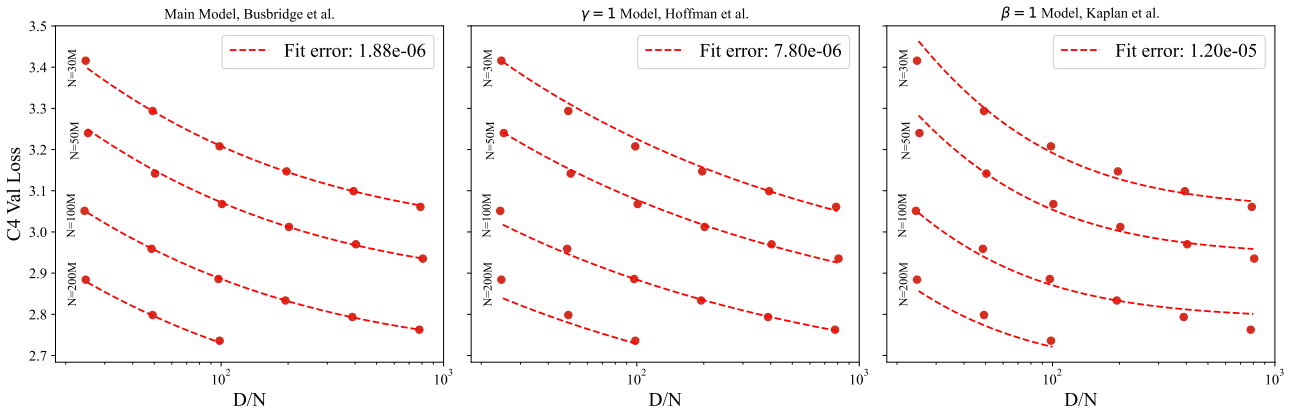

Figure 4. Comparison of various scaling law fits and their errors.

F. 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 eff_N and eff_D parameters using the same loss function.

For the isolated methods compared in Section 2, we fit eff_N and 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 $\gamma = 1$ [23] and $\beta = 1$ [24]. The fits are presented in Figure 4 alongside the mainly used of Busbridge et al. [8].

Parameter	A	α	B	β	E	γ
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.

G. Performance breakdown

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` [30].

The figure compares three kernel configurations. The left subplot shows our fused kernel for quantization-related operations using a basic 32×32 threadblock tile size. The center subplot increases this tile size to 128×32, 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%.

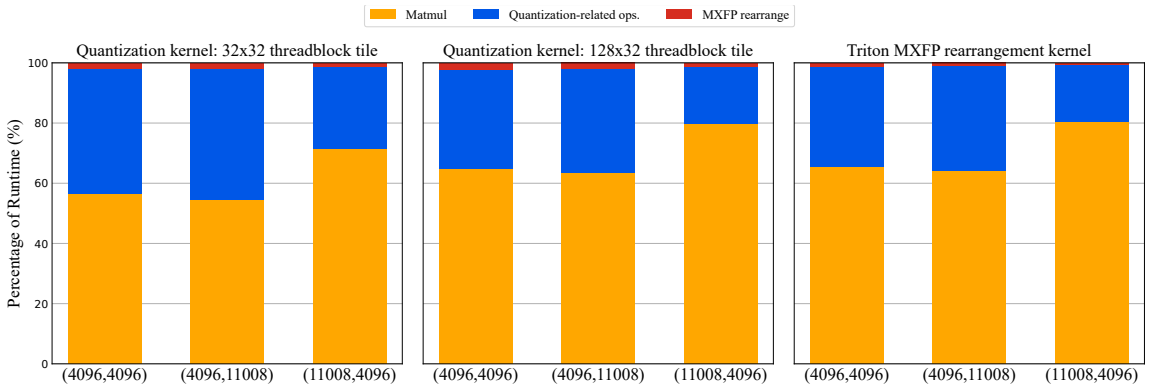

Figure 5. Breakdown of runtime composition across three linear layer shapes of a Llama-7B model, for an input of batch size 64, and sequence length 512.

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 [37]. The Triton backend exhibits similar trends, with rearrangement overheads further reduced and matrix multiplication dominating the total runtime.

H. End-to-end prefill speedups

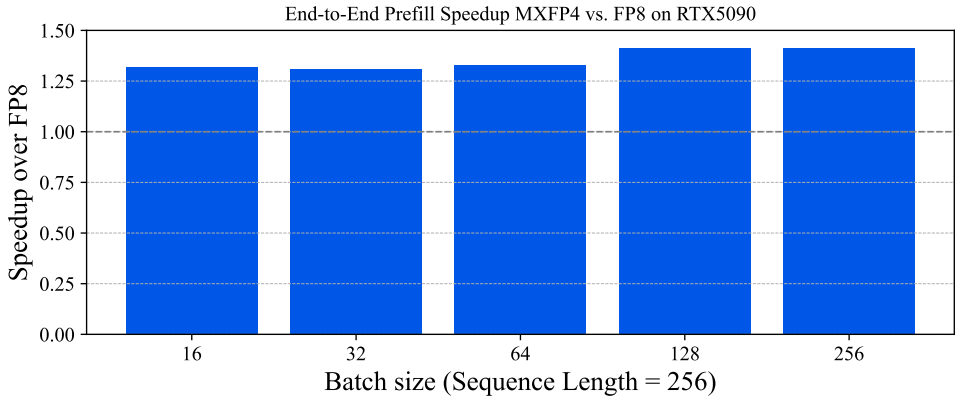

Figure 6. End-to-end prefill speedups for Quartet MXFP4 vs. FP8, across different batch sizes, using the 7B parameter model on a single RTX 5090.

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.

I. 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 [19]. To ensure a fair comparison, we introduce two key modifications to the original QUAROT approach:

1. **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.
2. **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.

J. 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.