TensorGauge: A Pre-silicon End-to-end Framework for Quantifying Numerical Effects of Tensor Core Microarchitecture in GEMM

Wei Liu[1], Yi Yu[2], Qianliang Liu[1], Xiaoyou Song[1], Mingyuan Ma[1],
Yuhan Wang[1], Haonan Sun[1], Yumin Hou[2], Hu He[1]
[1]School of Integrated Circuits, Tsinghua University, Beijing 100084, China.
[2]College of Computer Science, Beijing University of Technology, Beijing 100124, China.
Emails: {liuw24, liugl24, songxy24, mmy23, yh-w24, shn24}@mails.tsinghua.edu.cn; yiyu33129@gmail.com;
houyumin@bjut.edu.cn; hehu@tsinghua.edu.cn.

***Abstract*—General matrix multiplication (GEMM) is the core computational primitive in deep learning, and its hardware implementations vary widely across modern AI accelerators. Due to the non-associativity of floating-point arithmetic, microarchitectural choices in GEMM compute units induce systematic numerical differences under finite precision, arising from variations in accumulation structure, rounding, and reduction. However, the end-to-end impact of these hardware-level numerical behaviors on neural network training and inference remains poorly understood. This paper presents TensorGauge, a pre-silicon end-to-end framework for quantifying numerical effects of tensor core microarchitecture in GEMM. TensorGauge combines bit-exact numerical modeling with RTL-backed design variants and a PyTorch-based execution environment, enabling unified evaluation from arithmetic units to full neural networks. Our results suggest that model-level requirements on intermediate accumulation precision can differ from conclusions drawn using operator-level error alone. Across representative low-precision formats, we observe format-dependent stability thresholds, suggesting tiered accumulation-precision targets for tensor core microarchitecture design. TensorGauge provides quantitative guidance for the pre-silicon design of multi-precision tensor compute units.**

***Index Terms*—Matrix multiply-accumulate, Numerical Error Analysis, Neural Network, Pre-silicon evaluation, Bit-accurate simulation**

I. Introduction

General Matrix Multiplication (GEMM) is a fundamental computational primitive in AI training and inference: convolutions are commonly lowered to matrix multiplication, while Transformer attention and MLP blocks are dominated by GEMM-like operations [1]–[3]. To meet this demand, GPUs and AI accelerators devote substantial compute to matrix/tensor units [4]–[6] and increasingly rely on mixed- and low-precision formats such as BF16, FP8, NVFP4, and MXFP4 [7], [8].

Accordingly, modern general-purpose GPUs (GPGPUs) and specialized AI chips dedicate a large fraction of their compute capability and silicon budget to matrix/tensor acceleration units (tensor cores). In NVIDIA GPUs, from A100 to B200, FP16 dense tensor-core throughput is typically 16×–31× that of FP32 vector units, making tensor cores dominant compute engines for AI workloads. However, compared to the standard floating-point operations performed by vector units (such as multiplication, addition, and FMA operations), tensor cores handle more low-bit floating-point formats, such as BF16, FP8, and NVFP4/MXFP4 [8]. These formats are designed to accelerate matrix multiplication and support various new computational precisions, which often leads to more opaque numerical behavior. Unlike arithmetic over real numbers, floating-point operations do not satisfy the associative property. GEMM requires the accumulation and reduction of a large number of products along the k-dimension. Throughout this paper, k denotes the GEMM reduction dimension, whereas K denotes the hardware dot-product group size, i.e., the number of product terms reduced by one DPU/FDA/SDA/GFDA primitive. Under finite precision, using different reduction structures and accumulation orders on the same set of products results in different rounding and truncation errors, leading to varying outputs. These discrepancies may arise within a single dot product, as well as across parallel partitions. Thus, understanding GEMM's numerical differences requires considering both software and hardware-level reduction and arithmetic semantics.

Broadly speaking, the sources of GEMM numerical discrepancies can be categorized into two types: (1) Software-side accumulation order and parallel reduction/merge policy. On massively parallel systems like GPUs, tiling, scheduling, and parallel reductions change the traversal and reduction structure along k. In practice, split-K and Stream-K style kernels partition the k-dimension into chunks that are computed in parallel and then merged, yielding implementation-dependent accumulation orders [9]. (2) Hardware-side tensor core organization and arithmetic data paths. Different matrix units hardwire the structure of matrix multiplication, such as the grouping of partial sums in dot products, local accumulation and propagation in systolic arrays, or aggregation patterns in outer product arrays, thereby enforcing different associativity structures. Additionally, the rounding path, whether the addition of C is fused into the dot-product data path, and the policy for intermediate precision retention all reshape error distributions and biases.

To evaluate the impact of different GEMM computation

This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB4500101.

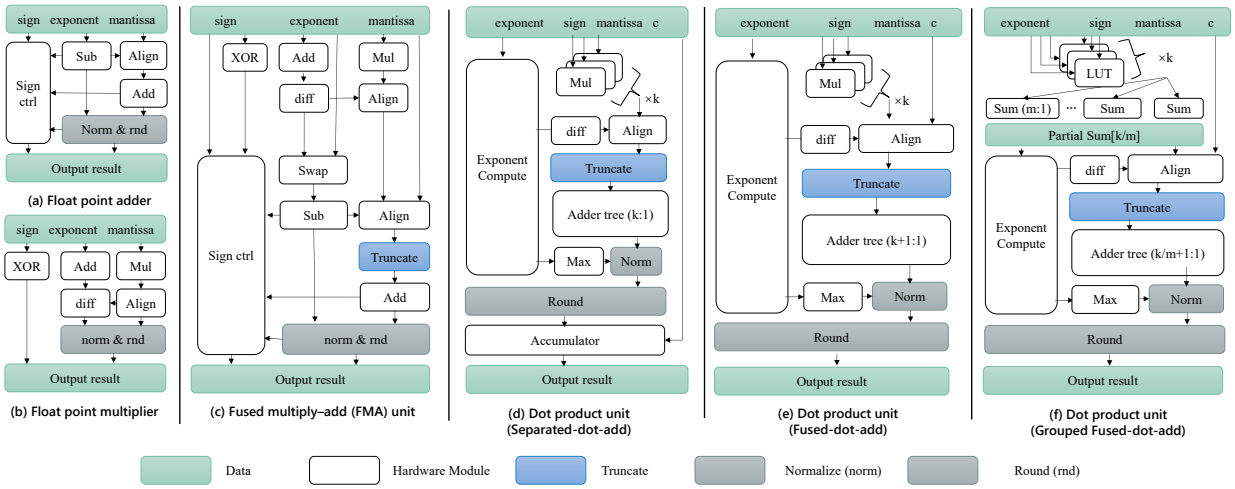


Fig. 1: Microarchitecture of floating-point operations across hardware units. (a)–(c) show scalar add, multiply, and fused multiply-add units; (d)–(f) show dot-product/add organizations that reduce four products before the optional addition of c.

methods, we propose TensorGauge: a pre-silicon, end-to-end framework designed to systematically characterize how tensor core design and reduction order impact neural network workloads. TensorGauge focuses on arithmetic paths directly related to numerical precision, while avoiding heavy simulation of irrelevant system components, thereby enabling controllable and scalable pre-silicon evaluation. Specifically, TensorGauge integrates: (1) a set of RTL/bit-level semantic variants of tensor cores (e.g., dot-product units, systolic array-style FMA organizations, and mul/adder stack structures), which explicitly control k-dimension accumulation structure, rounding paths, and intermediate accumulation precision retention policies; (2) a CUDA-accelerated bit-accurate numerical model, which can be plugged into PyTorch to replace the underlying GEMM numerical behavior, thereby reproducing different semantics during training and inference; (3) end-to-end workload execution to compare the impact of different design choices on real neural networks.

Using this framework, we systematically evaluate and address the impact of tensor core hardware design, numerical behavior, and end-to-end effects on neural networks. Our contributions are as follows:

- Controllable tensor core semantics and bit-accurate modeling: We implement and align multiple numerical semantic variants of tensor cores, explicitly covering GEMM numerical discrepancies induced by k-dimension reduction structure, rounding paths, and intermediate precision retention policies.
- Pre-silicon end-to-end evaluation framework: Considering that traditional CPU-based hardware simulators are slow and require complex software toolchains for integration into deep learning workloads, we have constructed a CUDA-accelerated bit-accurate model and integrated it into PyTorch, enabling end-to-end training and inference evaluation without the need for expensive full-system simulation.
- Using representative training and inference workloads, we derive hardware-facing provisioning guidelines for intermediate accumulation precision across multiple low-precision formats: we show that operator-level error metrics alone can be insufficient, and we identify format-dependent thresholds that motivate tiered design targets rather than a single one-size-fits-all precision.

TensorGauge is a pre-silicon framework for end-to-end numerical evaluation of tensor core semantics. By integrating at the PyTorch dispatch level via custom operators and transparent layer replacement, it can be readily applied to a broad range of models and workloads. The rest of this paper is organized as follows: Section II introduces the motivation for this work; Section III analyzes GEMM computation precision; Section IV details the TensorGauge framework and implementation; Section V describes the evaluation setup; Section VI reports experimental results and analysis; Section VII reviews related work; Section VIII concludes the paper.

II. Motivation

Floating-point GEMM can diverge across hardware even under the same datatype because IEEE 754 [10] specifies local rounding semantics but not the parallel reduction tree, merge order, or tensor-core datapath organization. TPUs [4], NVIDIA tensor cores [5], AMD matrix cores [11], and NPUs [6] therefore expose different numerical behavior, and even successive NVIDIA generations are not numerically identical [12].

Our focus is narrower than full-system simulation: we ask how changes in GEMM compute-unit semantics propagate to model-level outcomes. Reports around DeepSeek V3 [13] and the published SageAttention2 study [14] suggest that limited accumulation precision can already become a training bottleneck. TensorGauge is therefore designed to provide GEMM-semantics fidelity, propagate those arithmetic choices to end-to-end training/inference, and determine whether the resulting differences stay within seed-level variation or materially affect convergence and quality.

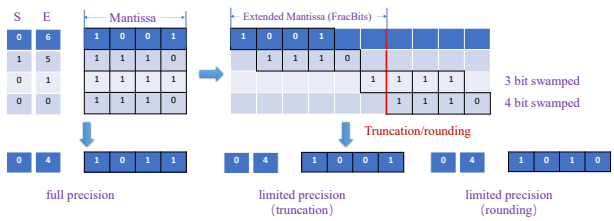


Fig. 2: Swamping in a 4-input aligned addition under limited intermediate precision. Small aligned terms can be shifted out of the retained datapath and become unrecoverable during later cancellation. The truncation and rounding examples use the same output width; rounding additionally keeps sticky information at the least significant bit (LSB), but still cannot recover values lost before accumulation.

III. Analysis of Floating-Point GEMM

With the transition of floating-point arithmetic from traditional ALUs and Fused Multiply-Add (FMA) units to emerging Dot Product Units (DPUs), the *swamping effect* [15] has shifted from a phenomenon observed across a sequence of distinct operations to one intrinsic to the internal behavior of a single hardware component. As illustrated in Fig. 2, insufficient intermediate precision causes smaller values to be shifted out or truncated, effectively excluding them from the final calculation. When subsequent operations involve the cancellation of large magnitude terms (catastrophic cancellation), the loss of these smaller values results in irreversible errors, a problem that persists even when rounding mechanisms are employed. In the context of DPUs, where multiple products must be aligned prior to accumulation, the risk of swamping errors due to insufficient intermediate bit-width is heightened. Consequently, achieving a balance between precision and efficiency by selecting an appropriate bit-width for intermediate products represents a critical challenge for DPU design.

In conventional FMA designs [16], only the addend requires shifting. When the input multiplier and addend share the same precision, the maximum shift width required is $3M + 4$, where M denotes the bit-width of the mantissa. Since left-shifting the addend beyond $M + 3$ bits in a single multiply-accumulate operation merely introduces zeros, hardware efficiency is improved by using logic signals to prevent shifting in meaningless zeros; the computation then relies on the product to calculate the sticky bit and concatenate it to the higher-order bits. Similarly, during right-shifting, if the addend shifts beyond the bit-width corresponding to the product, the shifted-out bits need only be reduced to a sticky bit via logical operations, eliminating the need for retention.

For a mixed-precision FMA where the input multiplier has a mantissa width of M_1 and the addend has M_2, assuming only the addend is shifted, the maximum requirements are $M_2 + 3$ bits for left shifts and $2M_1 + 1$ bits for right shifts. This results in a total maximum shift width of $M_2 + 2M_1 + 4$. Consequently, the adder width is no longer the $2M + 2$ bits typically cited in the literature, as the input multiplication precision differs from the accumulation precision in mixed-precision scenarios. In conventional designs, bits from the addend shifting out of the $2M + 2$ range are compressed into a sticky bit. However, when $M_1 \neq M_2$, the $2M_1 + 2$ bits of the product significand may not suffice to meet output precision requirements. Therefore, under mixed-precision conditions, the adder's bit-width must be sufficient to ensure the correct calculation of the sticky bit.

TABLE I: Intermediate-precision requirements for separated dot-add (SDA), fused dot-add (FDA), and representative NVIDIA Hopper/Blackwell tensor-core paths. The table contrasts theoretical lossless datapath widths with practical hardware choices; *NVFP4 uses the grouped fused dot-add (GFDA) organization.

Format	SDA	FDA	NV (Hop.)	NV (Blk.)
BF16 (E8M7)	522	522	25	25
FP16 (E5M10)	80	178	25	25
FP8 (E5M3)	66	164	13	25
FP4 (E2M1)	6	132	-	25*

When designing a DPU, achieving lossless precision cannot be accomplished simply by extending the single FMA maximum shift width of $3M + 4$ by $\lceil \log_2(K) \rceil$ bits, as suggested in [16]. Here, K denotes the number of multiplicative terms reduced inside one dot-product operation, rather than the full GEMM k dimension. The DPU processes K products per hardware dot-product group; if bits shifted out of the effective range are merely reduced to a sticky bit via an OR operation and subsequently discarded, the swamping effect may be introduced. While existing designs may saturate the left shift at $M_2 + 3$ (relying on the product for the sticky bit), multi-product scenarios require the precise recording of shift amounts. Failure to do so risks catastrophic cancellation, where the contribution of smaller terms becomes critical. Without accurate shift and addition handling for these smaller values, the correct result becomes unrecoverable.

Consequently, realizing a fully precise, error-free DPU necessitates the retention of all intermediate results [17]. Let E_{in} denote the exponent-bit width of the input format. Specifically, the lossless precision for the Separated Dot-add (SDA) is calculated as $(2^{E_{in}} - 2) + (2^{E_{in}} - 4) + 2(M + 1)$, while for the Fused Dot-add (FDA) aligning to an FP32 accumulator, it is derived as $(2^{E_{in}} - 2) + \max(2^7 - 2, 2^{E_{in}} - 4) + 2(M + 1)$. As presented in Table I, these theoretical intermediate precisions required to strictly avoid swamping effects are excessive across various input formats. These required bit-widths impose a prohibitive hardware overhead, rendering such a design physically impractical for implementation. Accordingly, implementations such as NVIDIA's Tensor Cores [18] and various academic studies [19] do not pursue full precision; instead, they employ truncation for values exceeding specific alignment ranges.

If retaining full intermediate precision is abandoned, is it theoretically possible to derive an implementation that sacrifices lossless accuracy while bounding the relative error for each operation? We argue that this is infeasible due to the stochastic nature of computational inputs, which inevitably leads to extreme corner cases. In typical AI workloads, data distributions vary significantly across neural network layers; shallow layers, in particular, frequently exhibit outliers that deviate from normal distributions [20]. Relying solely on

theoretical derivation makes it difficult to identify an optimal solution that omits full precision. Furthermore, there lacks a clear analytical path mapping the impact of a single dot-product operation to the overall model performance. Therefore, we posit that an end-to-end experimental framework is essential to analyze the problem through empirical data.

IV. TensorGauge Framework and Implementation

A. Overview

Fig. 3 provides an overview of TensorGauge, which unifies hardware specification, bit-accurate modeling, deep-learning execution, and statistical analysis into a single workflow. This enables rapid, fine-grained pre-silicon exploration of precision-related design choices and their end-to-end impact. The framework consists of four tightly coupled phases. **Phase 1** encodes hardware specifications and architectural knobs into a bit-accurate CUDA reference model that faithfully reproduces the target arithmetic behavior. **Phase 2** integrates the model into PyTorch via custom operators and layer-replacement mechanisms, enabling transparent substitution of standard compute kernels. **Phase 3** executes the resulting stack on workloads ranging from unit-level microbenchmarks to full end-to-end training and inference. **Phase 4** applies an analysis suite to extract numerical signatures and to study how errors scale with network depth and accumulation behavior. Finally, the analysis closes the loop by informing iterative refinements of hardware specifications and design parameters, yielding a continuous design-evaluate-optimize cycle.

B. Hardware RTL Implementation

In Phase 1, we implement a suite of hardware compute units in Chisel with configurable intermediate precision. The design space spans internal accumulation precision, operand formats, and quantization strategies, while keeping the surrounding architectural context fixed to isolate the effects introduced by the arithmetic unit itself. As illustrated in Fig. 1, we categorize the arithmetic micro-architectures according to (i) the accumulation organization along the reduction dimension and (ii) whether partial sums are grouped prior to the final reduction: (1) **Adder/multiplier**: a baseline implementation that realizes matrix multiplication by explicitly composing multipliers and adders, i.e., performing multiply and add as separate operations with intermediate rounding and/or format truncation at datapath-defined boundaries. (2) **FMA (fused multiply-add)**: a classic fused operation $a \times b + c$ with a single final rounding, typically retaining higher internal significand precision. This style is widely used in CPU floating-point pipelines and serves as a representative fused baseline. (3) **FDA (Fused Dot-Add)**: a fused reduction primitive that takes a K-term dot product and an additive scalar c simultaneously, i.e., $\sum_{i=1}^{K} a_i b_i + c$. This formulation aligns with the accumulation behavior commonly adopted by NVIDIA GPU Tensor Cores. (4) **SDA (Separated Dot-Add)**: a decoupled variant in which dot-product accumulation and the incorporation of c are separated, i.e., the dot product is reduced first and then combined with c in a subsequent stage. This organization is common in NPU designs, as it enables broader reuse of the accumulator and improves datapath flexibility. (5) **GFDA (Grouped Fused Dot-Add)**: when K is large, the reduction is performed hierarchically by partitioning terms into groups. Each group accumulates locally with higher precision, and the resulting partial sums are truncated/quantized before the final accumulation. This pattern is typical for LUT-based ultra-low-precision units, where FP4×FP4 products are first mapped to a higher-bit fixed-point representation via LUT, accumulated exactly within groups, and then aligned for the final reduction.

C. Sequential Accumulation Order

Floating-point reduction error is sensitive to summation order. To isolate accumulation effects, we evaluate several sequential schemes over product terms $\{t_i\}$, where $t_i = a_i b_i$: (i) **Best-case**: compensated summation in the style of Kahan [21] to mitigate truncation when adding small terms to a large partial sum; (ii) **Natural**: accumulation in the original traversal order as a realistic baseline; and (iii) **Grouped**: a two-level reduction that bins terms by magnitude (exponent and top mantissa bits), reduces within bins, and then reduces across bins, yielding a hierarchical summation structure. We additionally construct an **anti-Kahan** that, under an adversarial extreme ordering, persistently reinjects and accumulates rounding residuals to expose numerical fragility beyond order-only effects. This anti-Kahan variant is not a standard numerical algorithm and is not proposed as a practical summation algorithm; it is a deliberately adversarial stress test used only to upper-bound sensitivity.

D. Bit-Accurate CUDA Model and AI Workload Pipeline

Existing pre-silicon evaluation approaches face a fundamental trade-off between fidelity and practicality. Instruction-set simulators (ISS) can provide bit-level correctness but are prohibitively slow for modern AI workloads. FPGA-/emulation-based platforms (e.g., ZeBu) offer higher throughput, yet require a complete software enablement stack (runtime, drivers, RTL integration, and operator support), which is costly and typically available only in late design stages. TensorGauge therefore targets numerical fidelity of the GEMM compute unit rather than full-system timing fidelity. Since our goal is to isolate the numerical impact introduced by *compute-unit precision*, we intentionally exclude vector/normalization units and SFU-based activation accelerators. Accordingly, we replace only the matrix multiplication operator and use a bit-accurate CUDA implementation as the reference model. **Bit-level CUDA model.** We re-implement the GEMM operator on GPU and validate functional equivalence through three-way cross-validation among the bit-accurate CUDA model, NVIDIA GPU Tensor Core execution, and RTL implementation. Specifically, we invoke Tensor Core MMA through inlined PTX and use the same `mma` instruction variants as the target hardware across BF16, FP16, FP8, and FP4 (including MX/NV variants). Test shapes align with the MMA PTX shape (e.g., m16n8k16). Under an NVIDIA GeForce RTX 5060, our model matches native GPU outputs bit-for-bit over 1,000,000 randomized test cases, including ±0, subnormals, Inf, and NaNs. In addition, the CUDA model and the RTL produce bit-identical results, confirming consistency between the software bit model and hardware-level semantics. Therefore, the

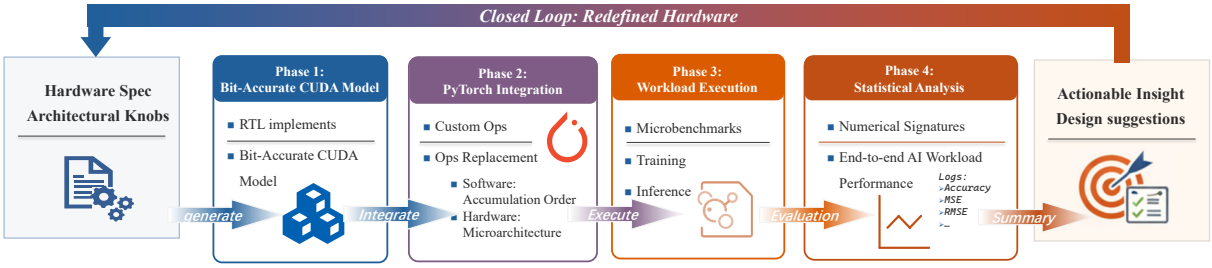


Fig. 3: TensorGauge workflow. Parameterized RTL and bit-exact CUDA models expose tensor-core semantic knobs, including reduction grouping, accumulation order, rounding, and intermediate precision; the PyTorch integration then propagates each hardware choice to unit-level metrics and end-to-end model quality.

simulator is faithful for the isolated GEMM semantics studied here, while memory-system behavior, control effects, and non-GEMM pipelines remain outside scope. **PyTorch integration.** The CUDA kernels are registered as custom operators and dispatched through the PyTorch dispatcher, enabling transparent replacement of standard matmul kernels with our bit-accurate variants.

V. Evaluation Setup

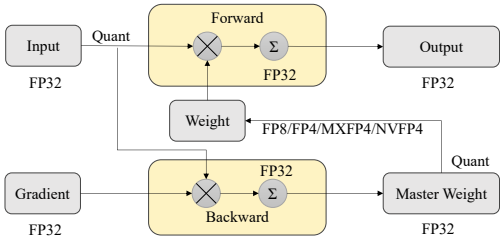


Fig. 4: Low-precision training recipe used in the end-to-end experiments. TensorGauge replaces GEMM semantics while keeping the optimizer, loss scaling, and model/training pipeline fixed, so differences can be attributed to arithmetic and reduction behavior.

The hardware design is written in Chisel and generates Verilog for each numerical configuration; synthesis is performed using Synopsys Design Compiler in the TSMC 22 nm process at a target frequency of 1.41 GHz. For power estimation, we additionally generate representative VCD traces by driving the RTL with normally distributed input operands, convert the traces to SAIF, and annotate switching activity in Design Compiler before reporting power. The reported PPA results remain pre-layout synthesis estimates used to compare trends across arithmetic variants; they do not claim sign-off accuracy after place-and-route or full-chip integration. For model-level simulation, while TensorGauge can be applied to many models with minimal engineering effort, in this paper we use a small set of representative training and inference case studies to validate the end-to-end methodology and extract hardware-facing insights, rather than aiming for exhaustive benchmark coverage. For training, we use BERT and evaluate multiple low-precision configurations, including FP16 mixed precision, FP8 (E4M3), and FP4/MXFP4/NVFP4. We follow a widely used low-precision training recipe (Fig. 4) : the forward pass is executed in low precision, while backpropagation and gradient accumulation remain in high precision. We evaluate on SST-2 from the GLUE benchmark and report the corresponding GLUE metric(s). For inference, we use Qwen3-0.6B [3] with BF16 computation, and report perplexity on WikiText-2. For training under different hardware settings, we run each configuration with five random seeds.

VI. Results and Analysis

A. Design unit evaluation

Hardware area and power. As shown in Fig. 5, we evaluate area and power using RTL generated by Chisel under a unified implementation and analysis flow. We implement and study three reusable compute units: (1) a BF16/FP16/TF32 unit, (2) an MXFP8/FP6/FP4 unit, and (3) an MXFP4/NVFP4 unit. For the BF16/FP16/TF32 path, the dataset also includes separated multiplier-plus-adder and FMA baselines across $K = \{2, 4, 8, 16, 32, 64\}$, so the comparison is not limited to dot-product primitives. To address activity dependence in power estimation, we regenerated the PPA dataset for all 232 RTL configurations using 512 cycles of normally distributed operands with a fixed random seed. The RTL simulations emit VCD traces, which are converted to SAIF and annotated in Design Compiler before power reporting. Thus, all PPA curves in Fig. 5 use activity-driven switching estimates rather than vectorless power defaults, while remaining pre-layout synthesis results. Fig. 5a and Fig. 5c report area and power efficiency as a function of the dot-product group size K, including MUL+ADD, FMA, FDA, and SDA curves and normalized to the FMA design at $K = 2$ (set to 100%). As K increases, both FDA and SDA exhibit improved area and power efficiency. In our experiments, area efficiency can increase by up to $\sim 135\%$, while power efficiency improves by up to $\sim 65\%$. In contrast, a chain of FMAs suffers from efficiency degradation due to the larger number of registers and pipeline stages, which introduce additional area and power overheads. We further examine the cost of intermediate fractional precision (FracBits) for the BF16/FP16/TF32-shared unit in Fig. 5b and Fig. 5d. For FDA, increasing FracBits by 1 bit incurs approximately a 1.70% area cost and a 1.64% power cost; for SDA, the corresponding costs are 1.43% and 1.40%. Moreover, the gap between SDA and FDA shrinks as FracBits increases: at FracBits $= 40$,

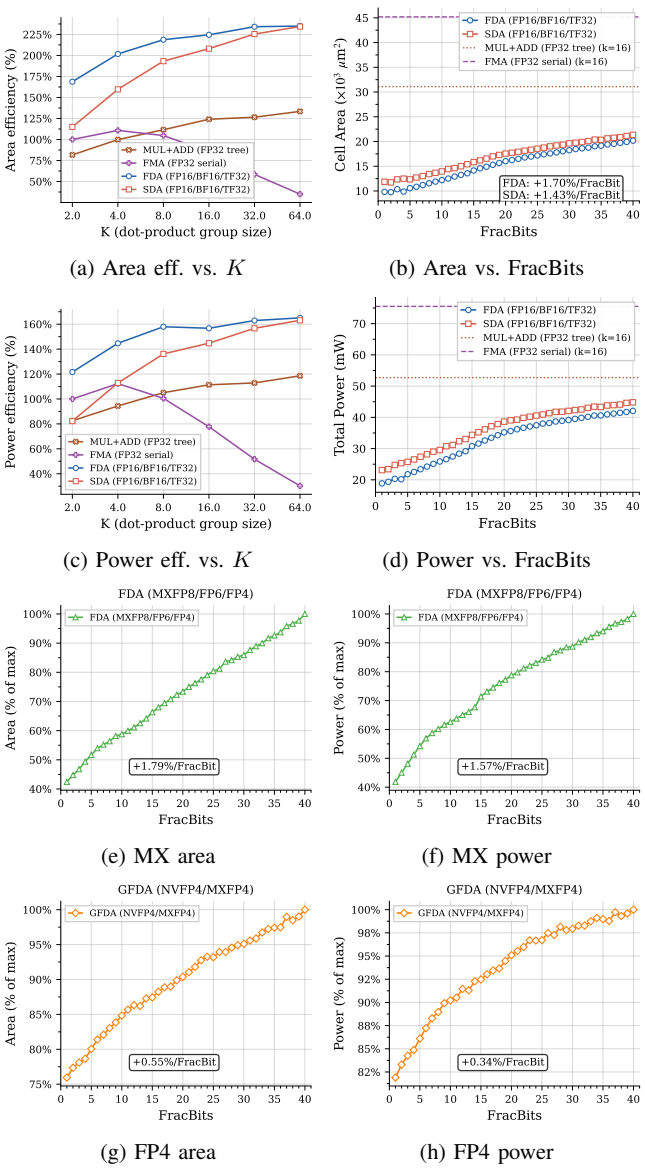
SDA consumes 5.78% more area and 6.53% more power than FDA; at FracBits $=$ 20, the overhead increases to 9.55% in area and 9.69% in power. This trend suggests that SDA and FDA become closer in implementation cost when higher intermediate precision is provisioned. Fig. 5e and Fig. 5f show the normalized area and power of the MXFP8/FP6/FP4-shared unit across different FracBits, which follows an approximately linear trend. For example, when FracBits increases from 20 to 40, the normalized area changes from 73.34% to 100%, and the normalized power changes from 78.76% to 100%. In terms of incremental scaling, each additional FracBits corresponds to about a 1.79% area cost and a 1.57% power cost. Similarly, Fig. 5g and Fig. 5h present the GFDA MXFP4/NVFP4 unit results, which exhibit consistent scaling: each extra FracBits introduces roughly a 0.55% area cost and a 0.34% power cost.

Arithmetic error characteristics. As shown in Fig. 6, we evaluate the numerical error characteristics of the compute unit. Following the statistical metrics in [22], we report the mean squared error (MSE) and the variance of squared error. In addition, we report the variance retention ratio (VRR) [15], which quantifies how much output variance is preserved relative to a high-precision reference under reduced-precision accumulation (VRR $=$ 1 indicates no variance loss). We generate 10000 test samples with inputs drawn from a normal distribution and compute the above statistics accordingly. Overall, increasing the intermediate fractional precision (FracBits) leads to rapid convergence of all error metrics, which become stable beyond certain FracBits thresholds. Specifically, Fig. 6a shows that the MSE stabilizes when FracBits $\geq$ 19, with magnitude below 10^{-8}. Fig. 6b reports that the variance of squared error drops below 10^{-9} when FracBits $\geq$ 17. Fig. 6c shows that VRR approaches 1 when FracBits $\geq$ 16, indicating that the accumulation largely preserves the second-order statistics and that variance loss due to swamping/rounding is negligible beyond this point. These results suggest that increasing intermediate precision suppresses rounding/truncation-induced errors and that the unit-level metrics enter a stable regime with relatively modest FracBits. In our setting, MSE and variance of squared error stabilize around FracBits $\approx$ 17–19, while VRR approaches 1 at FracBits $\geq$ 16. We note that model-level training/inference can impose stricter precision requirements than these unit-level stability points, motivating the end-to-end evaluation in Section VI-B.

B. End-to-end Neural Network results

TABLE II: Simulation throughput for GPT-2 decoding, reported in KIPS (kilo instructions per second).

Simulator/Platform	Speed (KIPS)
Multi2Sim [23]	0.8
Accel-Sim (exec-driven) [23]	6
Accel-Sim (trace-driven) [23]	12.5
MGPU-Sim [23]	28
Macsim [24]	50.5
FPGA for Ventus [25]	781.25
TensorGauge (Ours)	440640
Real GPU [24]	4103750



(a) Area eff. vs. K (b) Area vs. FracBits

(c) Power eff. vs. K (d) Power vs. FracBits

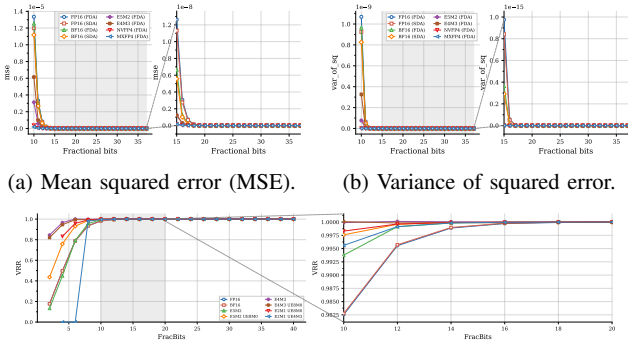
(e) MX area (f) MX power

(g) FP4 area (h) FP4 power

Fig. 5: Area and SAIF-annotated power trends for generated compute units. (a) and (c) sweep dot-product group size K for MUL+ADD, FMA, FDA, and SDA, with efficiency normalized to the $K = 2$ FMA baseline; (b) and (d) sweep FracBits at $K = 16$; (e)–(f) show normalized MXFP8/FP6/FP4 costs; (g)–(h) show normalized GFDA MXFP4/NVFP4 costs.

Simulation throughput. Table II reports the simulator throughput for GPT-2 decoding that generates 100 new tokens. Compared with a real GPU, the simulator achieves 10.74% of native GPU speed. Despite this slowdown, it is 4–5 orders of magnitude faster than conventional GPU simulators and 564.02× faster than an FPGA-based alternative, which makes it practical for end-to-end model-level experiments.

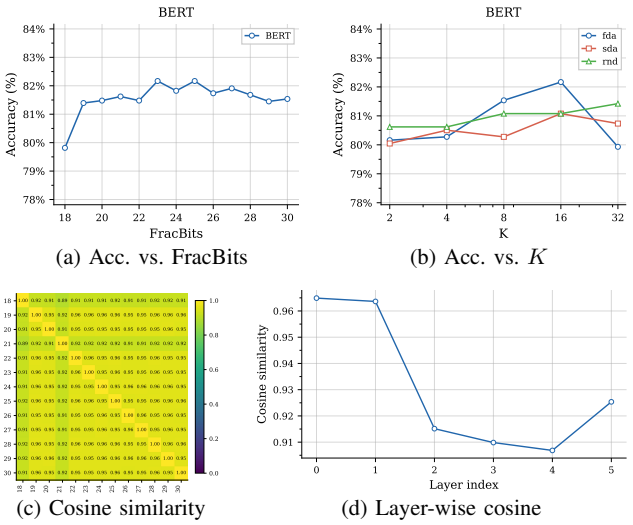
Training behavior under different hardware parameters. Fig. 7 summarizes the impact of hardware parameters on training accuracy and weight cosine similarity. As shown in



(a) Mean squared error (MSE). (b) Variance of squared error.

(c) variance retention ratio (VRR).

Fig. 6: Overall error statistics across configurations.

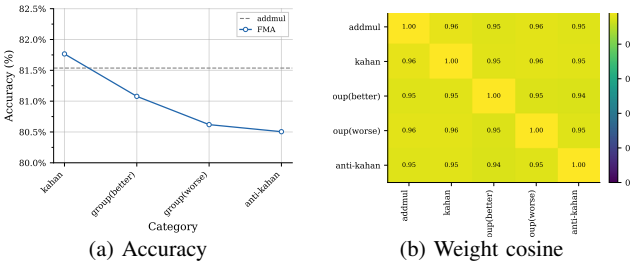


(a) Acc. vs. FracBits (b) Acc. vs. K

(c) Cosine similarity (d) Layer-wise cosine

Fig. 7: BERT training under different hardware semantics. The top row reports best accuracy versus FracBits and dot-product group size K; the bottom row compares final weights through pairwise and layer-wise cosine similarity, exposing trajectory divergence even when accuracy is similar.

Fig. 7b, across different K settings, both FDA and SDA achieve the best accuracy at $K = 16$; overall, neither design exhibits a clear advantage in end-to-end outcomes. With $K = 16$ fixed, Fig. 7a shows that mixed-precision FP16 training enters a stable regime when FracBits > 23, with only minor fluctuations. Beyond accuracy, the learned weights diverge across hardware settings. Fig. 7c indicates that pairwise cosine similarity differs across configurations, with more pronounced deviations at lower FracBits. Moreover, Fig. 7d shows that these discrepancies grow with network depth and further amplify as training progresses. This suggests that, as data and compute accumulate, hardware-induced numerical effects can steer optimization toward different local solutions. In our experiments, these local solutions are usually comparably acceptable once FracBits is above the stable regime, because the best-accuracy curves differ only mildly and stay close to seed-level variability; the issue becomes material in low-FracBits or adversarial settings, where the final accuracy gap

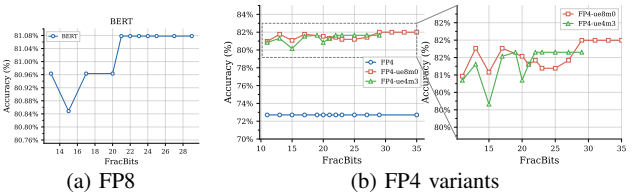


(a) Accuracy (b) Weight cosine

Fig. 8: Impact of accumulation order on BERT training. Accuracy and weight-cosine results compare best-case, random, natural, grouped, and adversarial anti-Kahan-style orders, isolating the effect of reduction order from datatype changes.

can exceed 1%.

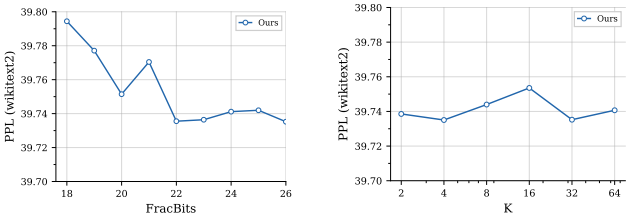
Impact of accumulation order. Fig. 8a shows that accumulation order leads to measurable accuracy differences: the best ordering achieves 81.76%, whereas a deliberately constructed worst ordering yields 80.50%, a gap of 1.26%. A random ordering achieves 81.53%, within 0.23% of the best case. Consistently, Fig. 8b reports pairwise cosine similarity of 0.94–0.96 across different orders, comparable to the similarity observed under FracBits$>$ 22 settings.



(a) FP8 (b) FP4 variants

Fig. 9: Low-bit BERT training accuracy versus intermediate FracBits. FP8 stabilizes near 21 bits, whereas FP4 formats require different margins, motivating format-dependent accumulator provisioning.

Low-bit training. For FP8 training, Fig. 9a shows that accuracy becomes stable once FracBits $= 21$. For FP4 training (Fig. 9b), we use direct quantization without finer-grained scaling; the limited dynamic range therefore yields lower end-to-end accuracy and weaker sensitivity to FracBits. Comparing FP4 variants, NVFP4 stabilizes when FracBits > 23, whereas MXFP4 requires FracBits > 28. Since MXFP4 and NVFP4 can share one hardware implementation, we recommend provisioning the higher precision target.



(a) PPL vs. FracBits (b) PPL vs. K

Fig. 10: Qwen3 BF16 inference sensitivity to hardware parameters. Perplexity remains nearly unchanged across FracBits and K, indicating that inference is less sensitive than training.

LLM inference. In Figs. 10a and 10b, for BF16 inference

on Qwen3, FracBits $= 22$ yields the lowest perplexity, and the perplexity difference across different K values is below 0.02.

VII. Related Work

Numerical non-associativity and determinism. Floating-point GEMM is not associative [26], so evaluation-order changes can yield different results. Prior work studies deterministic training [27] and the extent to which neural networks tolerate or even exploit numerical noise [28]–[30].

Numerical comparison methodology and tensor core modeling. A practical challenge is determining when two GEMM implementations are close enough. Elementwise tolerance checks are useful for regression testing but do not isolate systematic differences from reduction/merge structure or intermediate rounding points; error statistics provide a complementary signature-based view [22]. Recent work also characterizes or reproduces vendor tensor-core semantics through inference methodologies [31] and bit-accurate reference implementations such as MMA-sim [12]. From a formal-methods perspective, Valpey *et al.* model three tensor-core generations with SMT and automatically generate discriminating inputs [32]. These approaches mainly fit or reproduce existing hardware behavior. By contrast, TensorGauge exposes microarchitectural semantics as explicit control dimensions for pre-silicon exploration; quantization-only tools such as Pychop do not reproduce tensor-core execution semantics [33].

Low-bit training and accumulator precision. Low-bit datatypes reduce storage and bandwidth demands but make accumulator precision a first-order concern. DeepSeek V3 [13] highlights this tension in Hopper-class systems, while prior arithmetic designs widen aligners or accumulators to improve robustness [34], [35]. Low-precision training theory studies minimally sufficient precision and analyzes failure modes such as swamping [15]. TensorGauge complements this literature with a pre-silicon framework that links controllable GEMM semantics directly to training stability and inference quality.

VIII. Conclusion

TensorGauge enables pre-silicon, end-to-end quantification of GEMM numerical effects induced by tensor-core microarchitecture. Different arithmetic semantics can produce diverging optimization trajectories during training, so intermediate accumulation precision should be provisioned from model-level stability targets rather than operator-level error alone. The appropriate target is format dependent, making TensorGauge a practical vehicle for joint accuracy–PPA exploration.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, 2016, pp. 770–778.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," 2019.

[3] A. Yang *et al.*, "Qwen3 Technical Report," 2025, arXiv preprint arXiv:2505.09388.

[4] N. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles, C. Young, X. Zhou, Z. Zhou, and D. A. Patterson, "TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*. Orlando FL USA: ACM, 2023, pp. 1–14.

[5] "NVIDIA Blackwell Architecture Technical Overview," NVIDIA technical overview, 2024. [Online]. Available: https://resources.nvidia.com/en-us-blackwell-architecture

[6] H. Liao, J. Tu, J. Xia, H. Liu, X. Zhou, H. Yuan, and Y. Hu, "Ascend: A Scalable and Unified Architecture for Ubiquitous Deep Neural Network Computing : Industry Track Paper," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 789–801.

[7] S. Narang, G. Diamos, E. Elsen, P. Micikevicius, J. Alben, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh *et al.*, "Mixed precision training," in *Int. Conf. on Learning Representation*, 2017.

[8] B. D. Rouhani, N. Garegrat, T. Savell, A. More, K.-N. Han, R. Zhao, M. Hall, J. Klar, E. Chung, Y. Yu, M. Schulte, R. Wittig, I. Bratt, N. Stephens, J. Milanovic, J. Brothers, P. Dubey, M. Cornea, A. Heinecke, A. Rodriguez, M. Langhammer, S. Deng, M. Naumov, P. Micikevicius, M. Siu, and C. Verrilli, "OCP Microscaling Formats (MX) Specification," Open Compute Project (OCP) specification, 2023.

[9] M. Osama, D. Merrill, C. Cecka, M. Garland, and J. D. Owens, "Stream-K: Work-centric Parallel Decomposition for Dense Matrix-Matrix Multiplication on the GPU," 2023, arXiv preprint arXiv:2301.03598.

[10] IEEE, "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.

[11] Advanced Micro Devices, "RDNA4 Instruction Set Architecture: Reference Guide," AMD documentation, 2025. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/radeon-tech-docs/instruction-set-architectures/rdna4-instruction-set-architecture.pdf

[12] P. Xie, Y. Wang, F. Yang, and M. Yang, "MMA-Sim: Bit-Accurate Reference Model of Tensor Cores and Matrix Cores," 2025, arXiv preprint arXiv:2511.10909.

[13] C. Zhao, C. Deng, C. Ruan, D. Dai, H. Gao, J. Li, L. Zhang, P. Huang, S. Zhou, S. Ma, W. Liang, Y. He, Y. Wang, Y. Liu, and Y. Wei, "Insights into DeepSeek-V3: Scaling Challenges and Reflections on Hardware for AI Architectures," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*. Tokyo Japan: ACM, 2025, pp. 1731–1745.

[14] J. Zhang, H. Huang, P. Zhang, J. Wei, J. Zhu, and J. Chen, "SageAttention2: Efficient Attention with Thorough Outlier Smoothing and Per-thread INT4 Quantization," in *Proceedings of the 42nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, A. Singh, M. Fazel, D. Hsu, S. Lacoste-Julien, F. Berkenkamp, T. Maharaj, K. Wagstaff, and J. Zhu, Eds., vol. 267. PMLR, 2025, pp. 75 097–75 119.

[15] C. Sakr, N. Wang, C.-Y. Chen, J. Choi, A. Agrawal, N. Shanbhag, and K. Gopalakrishnan, "ACCUMULATION BIT-WIDTH SCALING FOR ULTRA- LOW PRECISION TRAINING OF DEEP NETWORKS," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[16] F. Niknia, Z. Wang, S. Liu, P. Reviriego, Z. Gao, P. Montuschi, and F. Lombardi, "A Configurable Floating-Point Fused Multiply-Add Design With Mixed Precision for AI Accelerators," *IEEE Transactions on Circuits and Systems for Artificial Intelligence*, vol. 2, no. 3, pp. 248–261, 2025.

[17] D. R. Lutz, A. Saini, M. Kroes, T. Elmer, and H. Valsaraju, "Fused FP8 4-Way Dot Product With Scaling and FP32 Accumulation," in *2024 IEEE 31st Symposium on Computer Arithmetic (ARITH)*. Malaga, Spain: IEEE, 2024, pp. 40–47.

[18] B. Hickmann and D. Bradford, "Experimental Analysis of Matrix Multiplication Functional Units," in *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*. Kyoto, Japan: IEEE, 2019, pp. 116–119.

[19] Y. Yao, C. Zhang, C. Qi, R. Chen, J. Wang, Z. Fu, N. Jing, X. Liang, and Z. Song, "SynGPU: Synergizing CUDA and Bit-Serial Tensor Cores for Vision Transformer Acceleration on GPU," in *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, 2025, pp. 1–7.

[20] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration," 2024, arXiv preprint arXiv:2306.00978.

[21] W. Kahan, "Pracniques: Further remarks on reducing truncation errors," *Communications of the ACM*, vol. 8, no. 1, p. 40, 1965.

[22] P. T. Peter Tang, "Rounding Error Statistics as Numerics Signature," in *2025 IEEE 32nd Symposium on Computer Arithmetic (ARITH)*, 2025, pp. 93–100.

[23] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. Valencia, Spain: IEEE, 2020, pp. 473–486.

[24] E. Chung, S. Na, and H. Kim, "Allegro: GPU Simulation Acceleration for Machine Learning Workloads," in *Machine Learning for Computer Architecture and Systems 2024 (Co-located with ISCA 2024)*, 2024.

[25] J. Li, F. Yu, M. Ma, W. Liu, Y. Wang, H. Wu, and H. He, "RISC-V-Based GPGPU With Vector Capabilities for High-Performance Computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 33, no. 8, pp. 2239–2251, 2025.

[26] O. Villa, D. Chavarria-Miranda, V. Gurumoorthi, A. Márquez, and S. Krishnamoorthy, "Effects of floating-point non-associativity on numerical computations on massively multithreaded systems," in *Proceedings of Cray User Group Meeting (CUG)*, vol. 3, 2009.

[27] D. Zhuang, X. Zhang, S. Song, and S. Hooker, "Randomness in neural network training: Characterizing the impact of tooling," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 316–336, 2022.

[28] Y. He, M. Hutton, S. Chan, R. De Gruijl, R. Govindaraju, N. Patil, and Y. Li, "Understanding and mitigating hardware failures in deep learning training systems," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3579371.3589105

[29] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," *Advances in neural information processing systems*, vol. 31, 2018.

[30] M. Srivastava, S. Arora, and D. Boneh, "Optimistic verifiable training by controlling hardware nondeterminism," *Advances in Neural Information Processing Systems*, 2024.

[31] F. A. Khattak and M. Mikaitis, "Accurate Models of NVIDIA Tensor Cores," 2025, arXiv preprint arXiv:2512.07004.

[32] L. Valpey, M. Lemerre, F. Pregaldiny, and A. Gotlieb, "An SMT Formalization of Mixed-Precision Matrix Multiplication: Modeling Three Generations of Tensor Cores," in *NASA Formal Methods*. Springer, 2025, pp. 302–319.

[33] E. Carson and X. Chen, "Pychop: Emulating Low-Precision Arithmetic in Numerical Methods and Neural Networks," 2025, arXiv preprint arXiv:2504.07835.

[34] H. Tan, J. Zhang, X. He, L. Huang, Y. Wang, and L. Xiao, "A low-cost floating-point fma unit supporting package operations for hpc-ai applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 71, no. 7, pp. 3488–3492, 2024.

[35] D. R. Lutz, A. Saini, M. Kroes, T. Elmer, and H. Valsaraju, "Fused fp8 4-way dot product with scaling and fp32 accumulation," in *2024 IEEE 31st Symposium on Computer Arithmetic (ARITH)*. IEEE, 2024, pp. 40–47.