Accelerate LLM training by using Tensor core and Mixed precision

Puhao Zhu
Puhao@kth.se

Jinye Gong
jinyeg@kth.se

Haoran Zhai
hzhai@kth.se

***Abstract*—This project speeds up a minimal CUDA GPT-2 training codebase in two steps. First, major GEMM operations are replaced with cuBLAS SGEMM, which lets cuBLAS choose highly optimized kernels and, on supported GPUs, automatically use Tensor Cores (e.g., TF32 paths) while keeping FP32 outputs. Second, the MLP's first fully connected layer is changed to a selective mixed-precision path using cublasGemmEx with FP16 inputs/weights and FP32 accumulation/output, followed by a bias add. We evaluate performance using per-step time and tokens/s (plus approximate TFLOPs/s), and check numerical impact by comparing final weights to the baseline using cosine similarity, relative L2, and max absolute error. Results show clear throughput gains from cuBLAS GEMM, with smaller additional speedup from the mixed-precision layer, while weight differences remain very small.**

***Keywords*—Tensor core, AI training, LLM, Nvidia, Mixed pricision**

I. CONTRIBUTION

Puhao Zhu: responsible for paper reading, propose the Methodology, finishing the corresponding parts of paper(introduction&Methodology) (33.3%)

Jinye Gong: responsible for experiment, visualize figures, finishing the corresponding parts of paper(Experiment setup) (33.3%)

Haoran Zhai: responsible for result analysis, finishing the paper(result&conclusion) (33.3%)

II. INTRODUCTION

In recent years, artificial intelligence (AI) technology has developed rapidly and has been widely applied in various fields such as image recognition, speech processing, and natural language understanding. As these applications continue to expand, the scale and complexity of AI models have also increased dramatically, especially in tasks like large-scale language models (LLM). Training these massive deep neural networks requires extremely high computational power and large-scale hardware resources. Traditional computational methods and hardware architectures often fall short in addressing these challenges. Therefore, improving computational efficiency during AI training has become a critical issue for both academia and industry.

AI training is a core component of AI development, typically relying on powerful hardware acceleration. In recent years, graphics processing units (GPUs) have become the standard platform for deep learning training, thanks to their exceptional parallel computing capabilities. GPUs significantly improve computational efficiency, especially when processing large-scale neural networks. The parallel structure of GPUs gives them an unparalleled advantage in matrix operations and high-dimensional data processing, making them indispensable hardware support for AI training. However, as model complexity and data volume continue to grow, traditional GPU acceleration methods still face performance bottlenecks in certain tasks, necessitating more efficient optimization solutions to meet the ever-increasing performance demands.

To address this issue, NVIDIA introduced Tensor Cores, hardware units specifically designed to accelerate matrix multiplication and other high-performance computation tasks, aiming to enhance GPU computational performance, especially in deep learning and high-performance computing. For example, the RTX 3060, based on NVIDIA's Ampere architecture, is equipped with 28 second-generation Tensor Cores, delivering up to over 100 TFLOPS of FP16 Tensor Core performance, which significantly enhances matrix computation efficiency. Tensor Cores perform matrix operations efficiently using lower precision formats (such as FP16 or INT8), dramatically increasing computation throughput while reducing memory bandwidth requirements. To further enhance training efficiency, mixed-precision computing has also been widely adopted in AI training. This technique combines low-precision computation (e.g., FP16) with high-precision computation (e.g., FP32), optimizing computational efficiency and performance while maintaining model accuracy by reducing computational workload and memory usage. Although Tensor Cores and mixed-precision technologies have achieved significant acceleration in many AI applications, the performance characteristics of Tensor Cores in AI training workloads are still not fully explored, which constitutes the focus of this study.

The main goal of this paper is to explore the acceleration effects of combining Tensor Cores with mixed precision on the AI training process, particularly in the context of large-scale language model (LLM) training. By systematically analyzing the advantages of Tensor Cores in accelerating matrix operations and combining them with mixed precision computing, we aim to validate the effectiveness of this combination in improving computational efficiency, reducing memory usage, and maintaining model accuracy. Our goal is not only to apply this technology but also to evaluate its performance in real-world

training tasks, providing new insights and methods for accelerating AI training.

The main contributions of this paper are as follows:

① Based on the open-source llm.c codebase, we develop an FP32 training implementation that supports NVIDIA Tensor Cores, leveraging cuBLAS for optimized matrix multiplication to improve computational performance;

② Building upon this foundation, we further introduce mixed-precision training using FP16 in the first fully connected layer, significantly improving training efficiency while preserving numerical stability and model accuracy;

③ Through extensive experiments, we systematically evaluate the acceleration benefits of combining Tensor Cores with mixed-precision techniques in large-scale language model training. In addition, by analyzing weight similarity between models trained with different precision settings, we demonstrate that the proposed optimizations have negligible impact on model capability, providing strong empirical evidence for their practical applicability.

The structure of this paper is as follows: The first section introduces the relevant background and research status, reviewing the development and application of Tensor Cores and mixed precision technologies; The second section details the implementation of mixed precision and Tensor Cores in large-scale language model training; The third section presents the experimental design, data analysis, and results discussion; Finally, the fourth section summarizes the research findings and presents future research directions.

III. Methodology

Based on the CUDA hardware profiling results, we observe that the GPU execution time is overwhelmingly dominated by computation rather than data movement, with approximately 99.5% of the total GPU time spent on CUDA kernel execution and only about 0.5% attributed to explicit memory operations such as cudaMemcpy and cudaMemset. Among all kernels, matmul_forward_kernel4 emerges as the primary contributor to GPU runtime, accounting for approximately 70.5% of the total execution time. This kernel implements the core matrix multiplication operations in the model, including those used in QKV projections and MLP layers, which are invoked frequently during each training iteration and involve substantial computational workload.

In comparison, highly optimized Tensor Core GEMM kernels generated by CUTLASS collectively account for only 18.7% of the GPU time, indicating significantly higher computational efficiency. This discrepancy suggests that matmul_forward_kernel4 may not fully exploit Tensor Core acceleration or may exhibit suboptimal computation and memory access patterns. Consequently, matmul_forward_kernel4 constitutes the primary performance bottleneck in the current training pipeline, and further performance optimization efforts should therefore focus on analyzing and improving the efficiency of this kernel.

This work adopts a ablation study to systematically investigate the impact of numerical precision and compute path selection on the training performance of GPT-2. Throughout all experiments, the model architecture, training procedure, and hyperparameters are kept identical. Only the numerical precision used in matrix multiplication operations and the underlying hardware execution path are varied, ensuring fairness and interpretability of the comparisons.

To decouple the effects of numerical precision and hardware execution path, the experiments are organized into three groups, forming a progressive comparison from no acceleration to full acceleration:

The first group serves as the baseline and employs a naïve implementation. All computations are performed using FP32 precision, and all matrix multiplications are executed on CUDA cores without utilizing Tensor Cores. This group provides a reference point for both training performance and numerical stability.

The second group represents a partially accelerated configuration. While the overall training procedure remains identical to the baseline, the matrix multiplication operations inside *matmul_forward_kernel4* are replaced by cuBLAS GEMM calls. This allows cuBLAS to implicitly leverage Tensor Core acceleration (e.g., via TF32 compute paths on supported GPUs), while keeping FP32 as the input and output data type. All other computations are still carried out in FP32 on CUDA cores. This configuration is designed to evaluate the performance impact of Tensor Core–enabled GEMM execution without explicitly introducing mixed-precision training.

The third group corresponds to a mixed-precision accelerated configuration. In this setting, the first fully connected layer implemented in matmul_forward_kernel4 is modified to use Tensor Cores combined with FP32 + FP16 mixed precision. The remaining parts of the training process remain unchanged. This group aims to evaluate the additional performance benefits introduced by mixed-precision execution on Tensor Cores, beyond pure Tensor Core acceleration, during GPT-2 training.

In all experimental groups, the FP32 + FP16 precision strategy is restricted to matrix multiplication operations. The overall training algorithm, optimizer, and parameter update procedure remain unchanged, preventing algorithmic variations from influencing the comparison.

Training throughput is used as the primary performance metric to quantify computational efficiency across different configurations. In addition, training loss trends are monitored to provide auxiliary insight into the impact of precision and compute path choices on training behavior.

Through this methodology, the proposed experimental design enables an isolated evaluation of the effects of Tensor Core utilization and FP32 + FP16 precision strategies on GPT-2 training performance, providing a clear foundation for subsequent experimental analysis.

IV. Experimental Setup

All experiments in this work are conducted based on Karpathy's llm.c framework, which provides a complete implementation of GPT-2 training. The framework features a lightweight and transparent code structure, allowing fine-grained control over numerical precision and matrix multiplication execution paths. This makes it particularly suitable for analyzing the impact of Tensor Core utilization and precision strategies on training performance.

The experimental setup was deployed on a high-performance mobile computing workstation equipped with an NVIDIA GeForce RTX 3060 Laptop GPU and an Intel-based host processor. The CPU is an 11th Gen Intel Core i7-11800H (Tiger Lake architecture), featuring 8 cores and 16 threads with a base frequency of 2.30 GHz, providing sufficient throughput for data preprocessing tasks.

The GPU relies on the NVIDIA Ampere architecture (silicon code GA106M), integrating 3,840 CUDA cores and 3rd Generation Tensor Cores, which offer native support for FP16 mixed-precision acceleration. Although the device is equipped with 6GB of GDDR6 video memory, the utilization of FP32/FP16 mixed-precision optimization allowed the system to accommodate all GPT-2 training configurations used in this study. The experiments were conducted without encountering Out-Of-Memory (OOM) errors or triggering system-level data swapping.

The experimental system runs on a Linux operating system. The GPU computing environment is based on NVIDIA CUDA, and matrix multiplication operations are dispatched through the cuBLAS library in order to trigger Tensor Core execution when applicable. All programs are compiled using NVIDIA's CUDA toolchain with standard optimization settings to ensure consistent performance across different experimental configurations.

For Experimental Groups:

Baseline: train_gpt2_fp32 (FP32, CUDA cores only)

Partial Tensor Core Acceleration: modified train_gpt2_fp32 where matrix multiplications in matmul_forward_kernel4 are dispatched via cuBLAS, enabling implicit Tensor Core acceleration (e.g., TF32 compute paths), while keeping FP32 inputs and outputs.

Tensor Core + Mixed Precision: modified train_gpt2_fp32 where the first fully connected layer in matmul_forward_kernel4 uses Tensor Cores with FP32 + FP16 mixed precision

Performance Metrics

Training throughput is selected as the primary performance metric to evaluate computational efficiency across different experimental configurations. Throughput is measured as the number of processed tokens per unit time, with the input token count treated as the independent variable to analyze performance behavior under varying workload sizes.

In addition to throughput, model weight similarity is also used as an evaluation metric to quantify the impact of different experimental configurations on model behavior. By comparing the similarity of model weights obtained under different numerical precision settings and Tensor Core utilization strategies, we assess whether the proposed optimizations affect the learned representations.

Across all experiments, the model architecture, training procedure, and training hyperparameters are kept identical, ensuring that any observed performance differences can be attributed solely to differences in numerical precision and Tensor Core utilization.

V. Result

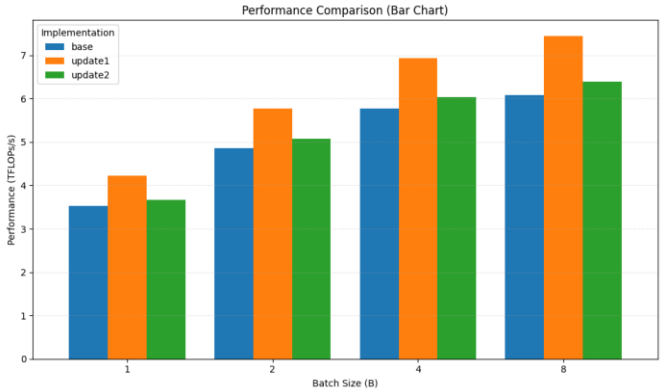


Figure X reports the throughput in TFLOP/s for the three implementations (base, update1, update2) as a function of the batch size B∈{1,2,4,8}. The throughput is computed from the number of floating-point operations in the GEMM kernels divided by the measured wall-clock time per training step. All measurements are taken on the same GPU and with identical model and optimizer settings; only the implementation of the matrix operations differs.

The base implementation uses hand-written FP32 CUDA kernels for all matrix multiplications and does not explicitly target Tensor Cores. Update1 replaces every GEMM in the training loop with a cuBLAS call. This allows cuBLAS to automatically route eligible operations to Tensor Cores while keeping FP32 as the only data type, so there is no change in numerical precision compared to the baseline. Update2 modifies only the first transformer layer. In this layer we explicitly call Tensor Core GEMMs and introduce mixed precision: FP16 is used for operations with lower accuracy requirements, whereas sensitive computations remain in FP32. All subsequent layers are kept identical to the base implementation.

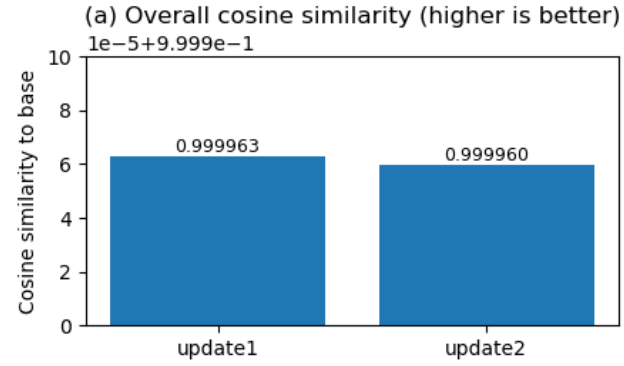
Across all three implementations, throughput increases monotonically with the batch size. For the base version, performance grows from roughly 3.5 TFLOP/s at B=1to about 6.1 TFLOP/s at B=8, reflecting better GPU utilization and amortization of kernel-launch overheads as B becomes larger.

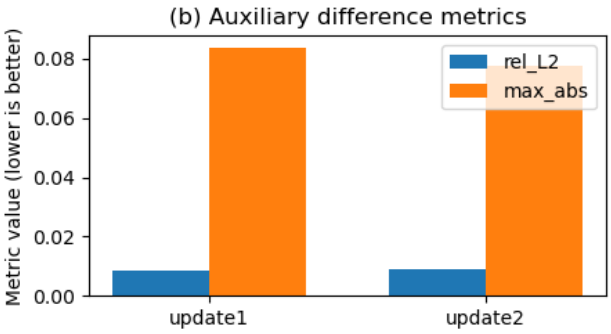
Update1 consistently achieves the highest throughput. Its values are approximately 4.2, 5.8, 6.9, and 7.4 TFLOP/s for B=1,2,4,8, which corresponds to an improvement of around 20% over the base implementation for every tested batch size. This shows that letting cuBLAS exploit Tensor Cores in all GEMMs has a clear and stable effect on performance.

Update2 also improves upon the baseline even though it optimizes only a single layer. Its throughput is typically about 5% higher than base but 10–15% lower than update1 at each batch size. This indicates that the first transformer layer accounts for a non-trivial fraction of the total compute cost, and that combining Tensor Cores with mixed precision in just this part of the model already yields a noticeable speedup. At the same time, the gap to update1 suggests that the remaining unoptimized layers still dominate a large part of the runtime. Overall, for all batch sizes we observe the same ordering of performance:

$$\text{update1} > \text{update2} > \text{base}$$

and no abnormal variance between runs, which confirms that the observed trends are robust.





To verify that the speed optimizations do not degrade model quality, we compare the trained weights of update1 and update2 against the baseline (base). We compute overall cosine similarity, relative L2 difference (rel_L2), and maximum absolute difference (max_abs) over all parameters. As shown in Figure Y, both update1 and update2 remain extremely close to the baseline, with cosine similarity above 0.99995 (i.e., nearly identical directions in parameter space). The auxiliary metrics are also small (rel_L2 around 0.9% and max_abs below 0.084), indicating that the numerical differences introduced by faster kernels and mixed precision are minor.

update1 tensor	rel_L2 (%)	update2 tensor	rel_L2 (%)
qkvb	2.009	qkvb	2.068
qkvw	1.056	ln2b	1.202
ln2b	1.041	qkvw	1.170
attprojw	0.898	fcw	0.998
fcw	0.891	attprojw	0.981

To understand where small differences concentrate, Table Y lists the tensors with the largest per-tensor rel_L2 values. For both updates, the largest deviations occur in attention-related parameters, especially QKV projection bias/weights (qkvb/qkvw), followed by ln2b. Importantly, these deviations remain at the 1–2% level for the most affected tensors, while the overall model similarity remains near-perfect. Together, the results support that the accelerated implementations achieve higher training speed without materially changing the learned model parameters.

VI. Conclusion

In this project, we compared three GPT-2 training implementations that differ only in how matrix operations are executed. The baseline (base) relies on hand-written FP32 CUDA kernels without explicit Tensor Core usage. We then designed, implemented, and tested two optimized variants. Update1 routes all GEMM operations through cuBLAS, enabling highly optimized kernels and allowing Tensor Cores to be used automatically when beneficial, while keeping computations in FP32. Update2 optimizes only the first transformer layer, but in that layer it combines Tensor Core GEMMs with mixed precision by using FP16 for numerically tolerant computations while retaining FP32 where higher precision is needed.

The performance results show a clear and consistent pattern. Switching from hand-written FP32 kernels to cuBLAS-based GEMMs (update1) yields an approximately 20% throughput improvement over the baseline across the tested batch sizes, even without changing numerical precision. Update2 is also faster than the baseline despite modifying only the first layer, indicating that the early transformer layer accounts for a meaningful fraction of the total compute and that selectively applying Tensor Core acceleration can still provide measurable speedups.

Crucially, the similarity-based quality check suggests that these speedups do not come at the cost of model quality. When comparing the final trained weights to the baseline, both update1 and update2 achieve near-perfect overall cosine similarity (> 0.99995), and the auxiliary difference metrics remain small (rel_L2 $\approx 0.9\%$ and max_abs < 0.084). Per-tensor analysis further shows that the largest differences are concentrated in attention-related parameters (especially QKV projections), while the overall parameter set remains extremely close to the baseline. Taken together, our results indicate that Tensor Core acceleration—especially when integrated via cuBLAS—can significantly improve training throughput while preserving the learned solution, and that mixed precision can further exploit this hardware when applied to numerically tolerant parts of the model.

Overall, the main takeaway from our project is that using Tensor Cores together with a careful mixed-precision design is an effective and practical way to accelerate GPT-2 training without materially altering the final trained parameters. A natural next step would be to extend mixed-precision Tensor Core acceleration beyond the first layer, and to validate model quality using downstream metrics (e.g., validation loss or task accuracy) in addition to weight similarity.

References

[1] *A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners," OpenAI, Tech. Rep., 2019. OpenAI*

[2] A. Vaswani et al., "Attention Is All You Need," in Advances in Neural Information Processing Systems (NeurIPS), 2017. arXiv

[3] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," arXiv preprint arXiv:1412.6980, 2014. arXiv

[4] S. Markidis, S. W. D. Chien, E. Laure, I. B. Peng and J. S. Vetter, "NVIDIA Tensor Core Programmability, Performance & Precision," 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Vancouver, BC, Canada, 2018, pp. 522-531, doi: 10.1109/IPDPSW.2018.00091.

[5] NVIDIA, "CUDA C++ Programming Guide," CUDA Toolkit Documentation, 2025. Accessed: Jan. 7, 2026.

[6] NVIDIA, "cuBLAS Math Modes (TF32 Tensor Core acceleration)," cuBLAS Documentation, 2023. Accessed: Jan. 7, 2026.