NestedFP: High-Performance, Memory-Efficient Dual-Precision Floating Point Support for LLMs

Haeun Lee, Omin Kwon, Yeonhong Park*, Jae W. Lee*
Department of Computer Science and Engineering
Seoul National University
{haeunlee, om0127, ilil96 jaewlee}@snu.ac.kr

Abstract

Meeting service-level objectives (SLOs) in Large Language Models (LLMs) serving is critical, but managing the high variability in load presents a significant challenge. Recent advancements in FP8 inference, backed by native hardware support, offer a potential solution: executing FP16 models by default, while switching to FP8 models during sudden load surges to achieve higher throughput at the cost of a slight quality degradation. Although this approach facilitates effective SLO management, it introduces additional memory overhead due to storing two versions of the same model. In response, this paper proposes NestedFP, an LLM serving technique that supports both FP16 and FP8 models in a memory-efficient manner by overlaying FP8 parameters onto FP16 parameters, allowing both models to share the same FP16 memory footprint. By leveraging a compact data format for the overlay and a specialized GEMM kernel optimized for this format, NestedFP ensures minimal degradation in both model quality and inference throughput across both FP8 and FP16 modes. NestedFP provides a flexible platform for dynamic, SLO-aware precision selection. The code is available at https://github.com/SNU-ARC/NestedFP.

1 Introduction

Large Language Models (LLMs) have emerged as foundational components in modern AI systems, enabling a wide range of applications such as virtual assistants, text and code generation, and multi-modal tasks that integrate textual and visual information [1, 8, 12]. Their broad applicability and strong performance have driven widespread adoption, leading to substantial user demand. For example, OpenAI's production-scale models process roughly 100 billion words per day [13] and production query rates frequently surpass 200 requests per second [18, 37].

To accommodate such large-scale applications, research efforts have focused not only on improving serving throughput [38, 17] but also on meeting service-level objectives (SLOs) [2, 29, 14]. One of the key challenges in achieving SLOs for LLM serving lies in handling dynamic load. Request rates, as well as input and output sequence lengths, can fluctuate significantly over short time intervals. Simply overprovisioning compute resources may mitigate this issue but results in substantial system cost inefficiency.

As a promising alternative, we explore an approach we call dual-precision LLM serving, which utilizes FP8 together with FP16, the de facto standard number format for weight parameters. While FP8 may incur a slight degradation in model quality, it offers up to 2× higher peak throughput compared to FP16 [22, 27, 28]. By operating in FP16 mode under normal conditions and switching to FP8 mode only during sudden load surges, we can effectively manage dynamic workloads without resource overprovisioning. This approach achieves a better balance between SLO attainment and service quality than relying solely on either FP16 or FP8.

*Corresponding authors

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

Co-deploying models in both formats, however, is non-trivial. Simply keeping both in memory is infeasible due to excessive memory capacity overhead. Storing only FP16 weights and quantizing them to FP8 on the fly when needed results in highly suboptimal FP8 throughput. Therefore, an effective solution is needed to support dual-precision LLM serving without incurring memory overhead or throughput degradation.

In this paper, we present NestedFP, a technique that enables efficient dynamic selection between FP8 and FP16 precision from a single unified model representation. NestedFP decomposes each 16-bit weight tensor into two 8-bit components, allowing FP8 inference without additional memory overhead or throughput degradation. Since FP16 mode requires reconstructing 16-bit values by merging the two 8-bit components in this approach, we develop a custom General Matrix Multiplication (GEMM) kernel, built on the CUTLASS library [24], that performs accurate on-the-fly FP16 reconstruction during kernel execution. According to our evaluation, NestedFP enables high-quality, dual-precision LLM serving efficiently: FP8 models generated from decomposed FP16 weights match the quality of existing FP8 quantization methods, and our FP16 GEMM kernel incurs only a 6.47% average performance gap relative to the CUTLASS baseline, which translates to 4.98% end-to-end overhead across models.

Our contributions are summarized as follows:

- We propose dual-precision LLM serving as an effective strategy to handle dynamic load fluctuations.
- We introduce a novel data format that supports both high-quality FP16 and FP8 inference in a memory-efficient manner.
- We develop a custom FP16 GEMM kernel for this data format, which performs on-the-fly FP16 value reconstruction during execution.
- We demonstrate that NestedFP enables high-quality dual-precision LLM serving with no memory overhead and high throughput, significantly improving SLO attainment compared to FP16-only baselines.

2 Background

2.1 Floating Point Representation

A floating point format with x exponent bits and y mantissa bits is denoted as ExMy. The exponent width x determines the dynamic range of representable values, while the mantissa width y governs numerical precision. A floating point value X_{FP} in this format can be expressed as shown in Equation 1, where S is the sign bit, M_j are the mantissa bits, E is the unsigned integer formed by the exponent bits, and b is the exponent bias.

$$X_{FP} = (-1)^S \cdot \left(1 + \sum_{j=1}^{y} M_j \cdot 2^{-j}\right) \cdot 2^{E-b}, \quad E \in \{0, 1, \ldots, 2^x - 1\} \tag{1}$$

Commonly used 16-bit formats include FP16 (E5M10) and BF16 (E8M7), while prevalent 8-bit formats include E4M3 and E5M2. These formats are natively supported by many modern hardware platforms and have become standard choices in low-precision machine learning computations [15].

2.2 FP8 Quantization for LLMs

Rise of FP8 Quantization for LLMs. While FP16 has become the de facto standard for LLM serving, 8-bit floating-point formats (FP8) are gaining traction for scenarios demanding even greater efficiency. This trend is driven by two main factors: (1) the representational advantages of floating-point formats, and (2) the increasing hardware support for FP8 arithmetic.

First, floating-point quantization is generally better suited for LLMs than integer quantization due to its wider dynamic range. Unlike in computer vision tasks—where integer quantization is prevalent—LLMs often exhibit activation outliers amplified by operations such as LayerNorm [28]. These outliers can significantly degrade model performance if not properly managed [15]. Although recent work has attempted to mitigate this issue in integer quantization through static outlier-aware techniques [28, 36], these methods still underperform compared to floating-point quantization, which inherently better accommodates long-tailed distributions and high dynamic ranges [21, 22, 40].

Second, FP8 support is rapidly being adopted by modern hardware accelerators, including NVIDIA Hopper GPUs and Intel Gaudi HPUs [15]. Leveraging this support, FP8 GEMM operations can achieve up to $2\times$ speedup compared to their FP16 counterparts. This growing availability is accelerating the adoption of floating-point quantization—particularly FP8—further highlighting its relevance for efficient LLM deployment.

E4M3 Format. Among the two widely adopted FP8 formats—E4M3 and E5M2—existing studies have consistently shown that E4M3 yields higher inference accuracy for Transformer-based language models [22, 28, 31]. However, due to its reduced dynamic range compared to FP16, the effectiveness of E4M3 critically depends on the application of appropriate scaling strategies [16, 22, 31]. Scaling is applied to align the dynamic range of tensor values with the numerical bounds of the E4M3 format. Scaling factors may be determined statically—based on heuristics such as the absolute maximum, percentile clipping, or mean squared error (MSE) minimization [22]—or computed dynamically during runtime. Weight tensors are typically scaled statically on a per-channel basis [21, 28, 40], most commonly using the absolute maximum value. Activation tensors, by contrast, may be scaled offline on a per-tensor basis [21, 28, 40] or dynamically on a per-token basis during inference. While the latter approach can improve accuracy, it incurs additional runtime cost due to the overhead of computing scaling factors on-the-fly [15, 27].

FP8 Results. While FP8 inference offers significant throughput gains, it comes with the potential caveat of quality degradation due to the inevitable loss of information associated with reduced precision [39]. Table 1 presents a comparison between FP8 and FP16 across various models and benchmarks. Specifically, for FP8 quantization, we use the E4M3 format with per-token activation and per-channel weight quantization for all linear layers, using the vLLM framework [33]. A consistent accuracy degradation is observed in FP8 models, which clearly shows the trade-off between throughput and model quality in FP8 quantization.

	Llama 3.1 8B			Mistral Nemo			Phi-4			Mistral Small		
	FP16	FP8	Δ	FP16	FP8	Δ	FP16	FP8	Δ	FP16	FP8	Δ
Minerva Math	18.2	17.6	-0.60	17.2	16.5	-0.72	42.7	42.9	+0.14	35.4	34.2	-1.14
MMLU Pro	33.3	32.8	-0.49	35.4	35.3	-0.06	53.1	52.7	-0.36	54.3	53.6	-0.77
BBH	39.5	38.6	-0.86	41.6	40.6	-0.97	27.5	27.1	-0.40	52.4	50.8	-1.62

Table 1: Downstream task accuracy for FP8 quantized models relative to FP16 models.

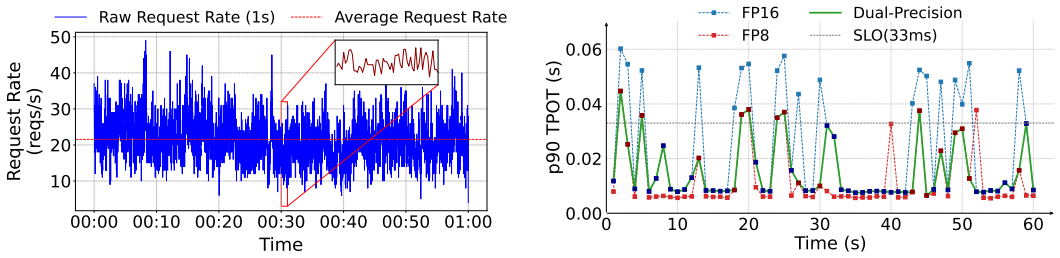
3 Motivation

3.1 Challenge: Dynamic Load in LLM Serving

LLM serving is characterized by substantial variability in request rates, as well as input prompt and output sequence lengths across requests [20, 30]. These variability is particularly problematic because the batch size scheduled at each iteration can fluctuate significantly, directly affecting the large deviations in Time To First Token (TTFT) and Time Per Output Token (TPOT). Such fluctuations occur in a highly dynamic and often unpredictable manner [29], making it difficult for service providers to consistently meet SLOs.

To quantify the extent of this variability, we analyze production traces from Microsoft Azure's LLM inference service [6]. Specifically, we measure how the request rate fluctuates over a one-hour interval (00:00–01:00 UTC, May 10, 2024, in the trace). Figure 1a presents the results. The request rate exhibits nearly a fivefold variation between the lowest and highest load periods. Moreover, as shown in the one-minute zoom-in view, these fluctuations occur at high temporal frequency, indicating second-level variability.

A classic approach to handling such load variations is auto-scaling, which adaptively adjusts computational resources (e.g., increasing or decreasing the number of GPUs in a cluster) [9]. However, auto-scaling is primarily effective in cloud environments with abundant resources and typically operates on minute-to-hour timescales [10]. Consequently, it is inadequate for LLM serving systems, which must respond to sharp, second-level load spikes and scale throughput within millisecond-level intervals to meet SLO requirements.



(a) (b)

Figure 1: (a) Request rate fluctuations during the first hour of May 10, 2024, from the Azure LLM inference trace. The inset plot highlights 1-minute variability from the 30-minute mark. (b) Comparison of p90 TPOT across different model precisions, using the Azure LLM inference trace recorded on May 10, 2024.

3.2 Our Proposal: Dual-Precision LLM Serving

To address short-term fluctuations in resource demand during LLM serving, we propose a simple yet effective dual-precision strategy that dynamically switches between FP16 and FP8 modes. When the system load is low, FP16 is used to preserve maximum model quality. Under high load conditions, the system falls back to FP8 mode to increase throughput, thereby maintaining responsiveness. Although FP8 mode may incur slight accuracy degradation, it is often preferable to violating SLOs during periods of request surges.

Figure 1b presents the per-second p90 TPOT under three precision schemes: FP16, FP8, and the proposed dual-precision format, using the Llama 3.1 8B model on an H100 GPU with vLLM. The request patterns are extracted from a 60-second segment of Microsoft Azure's LLM inference trace [6], scaled down to 20% of the original load. The TPOT SLO threshold is set to 33 ms [7]. As shown, the FP16 model exhibits more frequent TPOT spikes that exceed the TPOT SLO threshold compared to the FP8 model. To further quantify the difference, we analyze the rate of SLO violations across the same trace: 35.8% of requests under FP16 exceeded the threshold, whereas FP8 reduced this to 17.3%. In short, the proposed dual-precision inference can achieve FP8-level SLO compliance while minimizing quality degradation.

3.3 How to Implement Dual-Precision LLM Serving?

To enable dynamic switching between FP16 and FP8 modes in LLM serving, two straightforward implementation strategies can be considered: separate storage and on-the-fly dequantization.

Separate Storage. One simple approach is to maintain both FP16 and FP8 models in memory and switch between them according to system load fluctuations. While this enables fast switching, it introduces a 50% memory overhead—a significant burden for LLM serving systems already constrained by model size. This memory capacity overhead not only limits the maximum deployable model size but also reduces the available memory for key-value (KV) caching, thereby restricting the number of concurrent requests and ultimately degrading overall serving throughput.

On-the-fly Quantization. An alternative approach is to store only the FP16 model and perform on-the-fly quantization to perform computation in FP8. Unlike the separate storage strategy, this approach avoids additional memory capacity overhead. However, its FP8 performance is suboptimal; although computation is carried out by FP8 units with high throughput, the full 16-bit weights must still be loaded from memory, effectively doubling memory traffic compared to native FP8 execution. As a result, memory bandwidth becomes a bottleneck, limiting the achievable throughput and diminishing much of FP8's advantage over the FP16 baseline.

These limitations highlight the need for a more sophisticated dual-precision serving mechanism—one that eliminates additional memory overhead while enabling efficient GEMM execution in both FP8 and FP16 modes.

4 NestedFP

Overview. Figure 2 illustrates the core concept of NestedFP, which stores only 16-bit weights in memory and extracts 8-bit weights from them when operating in FP8 mode. Specifically, NestedFP decomposes each FP16 parameter into two parts: the upper 8 bits and the lower 8 bits. With these two parts of the weight parameters stored as separate tensors, both parts are loaded in FP16 mode, whereas only the upper part is loaded in FP8 mode. This approach not only eliminates additional memory usage but also prevents memory traffic amplification—for example, fetching 16-bit weights when executing in 8-bit mode.

Challenge 1: 8-bit Mode Accuracy. One key challenge of this approach is ensuring that the 8-bit representation extracted from 16-bit parameters achieves sufficient accuracy. In other words, it must maintain performance comparable to existing FP8 quantization methods. Naive truncation (simply using the upper 8 bits of FP16 values) yields a representation similar to E5M2, which offers limited precision compared to the commonly preferred E4M3 format for LLM inference. Moreover, such truncation often performs worse due to indiscriminate rounding, further degrading accuracy. This motivates the need for a quantization-aware data format that enables the extraction of high-quality FP8 representations. Section 4.1 elaborates on this data format design.

Challenge 2: 16-bit Mode Throughput. Another key challenge is maintaining throughput in the 16-bit mode. The 8-bit mode in NestedFP trivially achieves high performance, as native FP8 GEMM kernels can be used directly after loading the upper-part tensor. However, the 16-bit mode is less straightforward: both the upper and lower-part tensors must be loaded and combined to reconstruct the original 16-bit weights. This requires a custom GEMM kernel capable of performing the reconstruction dynamically. Section 4.2 details this kernel design.

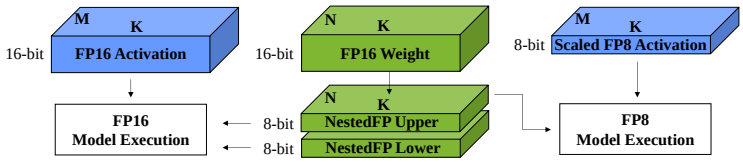
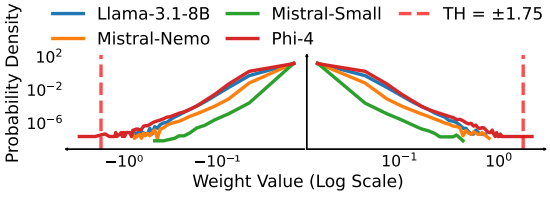


Figure 2: FP16 weight is decomposed into 8-bit upper and lower parts. Both parts are used for FP16 mode while only upper part is used for FP8 mode.

4.1 Compact yet Effective Data Format for Dual-Precision LLM

Opportunities: Low-Entropy Exponent Bit in FP16 Models. Figure 3a shows the the weight distributions of all FP16 linear layers across four different LLMs. The vast majority of weight values have absolute magnitudes less than or equal to 1.75, which means that their most significant exponent bits in FP16 are zero. Specifically, as shown in Figure 3b, three of the four evaluated models exhibit this trait across all layers. The sole exception, Phi-4, contains some layers that deviate from this pattern, but these account for only 8.75% of its layers. This strong tendency suggests an opportunity for lossless exponent mapping from FP16 to the FP8 E4M3 format, which also uses 4 exponent bits.



(a)

	Min	Max	NestedFP Applicable Layers
Llama 3.1 (8B)	-0.92	0.91	128 of 128 (100%)
Mistral Nemo Base (12B)	-0.92	0.95	160 of 160 (100%)
Phi-4 (14B)	-3.60	3.89	146 of 160 (91.25%)
Mistral Small Base (24B)	-0.49	0.45	160 of 160 (100%)

(b)

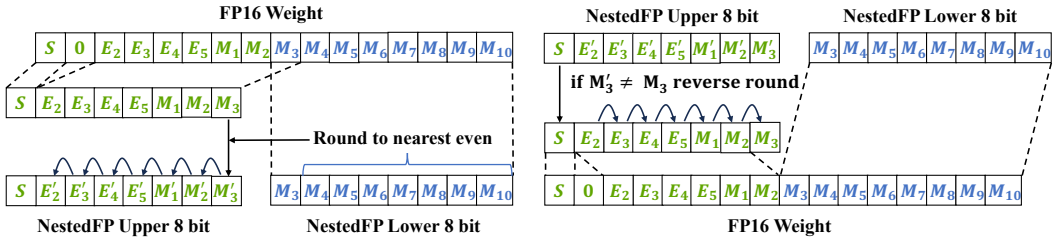
Figure 3: (a) Weight distributions of linear layers across different models. (b) Per-model weight range (min/max) and the proportion of linear layers where NestedFP can be applied.

Our Proposal. Figure 4a illustrates how NestedFP decomposes an FP16 weight such that the upper 8 bits can serve as a high-quality E4M3 representation by leveraging an unused exponent bit in FP16. First, the lower 8 bits directly inherit the lower 8 bits of the FP16 mantissa. The upper 8 bits, which

correspond to the E4M3 representation, are constructed as follows. The sign bit is directly inherited from FP16. The next four exponent bits are also copied from FP16, excluding the most significant exponent bit, which is assumed to be unused (or zero). Finally, 3 most significant mantissa (i.e., bits M[1:3]) bits are obtained and constructed value is applied rounding using a round-to-nearest-even policy. Specifically, the 7 least significant bits of the FP16 mantissa (i.e., bits M[4:10]) are examined to determine whether the value lies above or below the midpoint (64). If the value is greater than 64, it is rounded up; if it equals 64, it is rounded up only when the least significant bit of the resulting 3-bit mantissa (M3) is 1; otherwise, it is rounded down. This decomposition process is performed offline prior to model execution.

In this scheme, removing most significant exponent bit effectively acts as multiplying 2^8 between E4M3 and FP16 value. In other words, NestedFP effectively applies E4M3 quantization with a fixed scaling factor of 2^8.

FP16 Reconstruction. Figure 4b illustrates how NestedFP reconstructs FP16 weights at runtime. Essentially, this process merges the upper and lower 8-bit parts, but not through a simple concatenation. The rounding decision made during decomposition must be reversed by checking an implicit checksum; if the LSB of the upper 8 bits differs from the MSB of the lower 8 bits, a rounding-down correction is applied before recombining the exponent and mantissa bits into a valid FP16 value. This process ensures precise, lossless recovery of the original FP16 weights.



(a) Refactoring to NestedFP

(b) Reconstruction of FP16

Figure 4: (a) Decomposition of FP16 weight values into two 8-bit parts (offline). (b) Reconstruction of FP16 weight values (online).

Handling Exception Layers. For layers containing weight values with magnitudes exceeding 1.75 (i.e., where the MSB of the exponent is non-zero), we retain them in FP16 and always perform FP16 computation. This may reduce the benefits of dual-precision LLM serving, as the throughput advantage of FP8 mode becomes less pronounced. However, it is important to recall that, as shown in our analysis of popular models (Figure 3), such layers are rare and therefore have minimal impact on the overall effectiveness of the dual-precision scheme.

4.2 FP16 GEMM Kernel with On-the-fly Reconstruction

Figure 5 illustrates the high-level execution flow of the NestedFP FP16 GEMM kernel compared to a standard FP16 GEMM kernel. We assume an NVIDIA GPU environment, where the CUTLASS implementation serves as the standard FP16 GEMM kernel, upon which our kernel is also built. In the standard FP16 GEMM kernel, weights are loaded into shared memory, transferred to registers, and directly consumed by compute units (e.g., tensor cores in NVIDIA GPUs) without any transformation. In contrast, the FP16 GEMM in NestedFP requires a reconstruction step before tensor core execution; weights are stored as two 8-bit tensors, which must be loaded into shared memory, copied to registers, and then reconstructed into full FP16 values. Only after this reconstruction can the data be used by the tensor cores.

Not to make this additional step of reconstruction introduce a bottleneck, we present two key optimizations: (1) merged bitwise operations and (2) a three-stage pipeline.

Efficient FP16 Reconstruction via Merged Bitwise Operations. FP16 reconstruction involves a large number of 8-bit bitwise operations. To reduce the associated overhead, we fuse four 8-bit operations into a single 32-bit instruction, following the approach in [26, 35]. Algorithm 1 illustrates how FP16 reconstruction is performed using merged bitwise operations. Conceptually, the process iterates over the weight tensor, where each iteration processes four elements. In each iteration, four

8-bit values are loaded from the upper and lower tensors (W_{upper} and W_{lower}) into 32-bit registers `u` and `l`, respectively (Line 5–6). Then, the sign bits are extracted from `u` and stored in another register `s` (Line 7). Next, the algorithm checks whether rounding has occurred by examining `l` (Line 8); if rounding is detected, the effect is reversed, thereby recovering the exponent and mantissa parts of the upper 8 bits (Line 9). Then, the original upper 8 bits, `u_orig`, are fully reconstructed by concatenating them with `s` (Line 10). Finally, `u_orig` and `l` are fused using NVIDIA's `__byte_perm` instruction (lines 11–12), completing the reconstruction. Note that all the above operations are simultaneously applied to four 8-bit values through a 32-bit instruction, effectively reducing the total number of bitwise operations required.

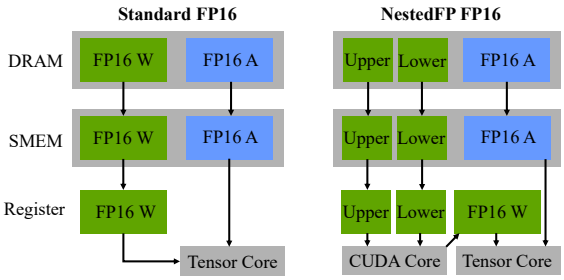


Figure 5: FP16 execution flow comparison.

Algorithm 1 FP16 Weight Reconstruction

1 $W_{\text{upper}}, W_{\text{lower}}$: upper and lower 8-bit tensor
2 W: reconstructed FP16 weight tensor
3 N: total number of elements in the weight tensor
4 **for** $i = 0$ **to** N **step** 4 **do**
5 `uint32_t u` $= W_{\text{upper}}[i : i+4]$
6 `uint32_t l` $= W_{\text{lower}}[i : i+4]$
7 `uint32_t s` $= u$ & `0x80808080`
8 `uint32_t sub` $= (l$ & `0x80808080`$) \gg 7$
9 `uint32_t u_orig` $= (((u - sub) \gg 1)$ & `0x3f3f3f3f`$)$
10 `u_orig` $=$ `u_orig` | `s`
11 $W[i : i+2] = \_\_byte\_perm($`u_orig`$, l,$ `0x1504`$)$
12 $W[i+2 : i+4] = \_\_byte\_perm($`u_orig`$, l,$ `0x3726`$)$
13 **end for**

Three-Stage Pipeline. The CUTLASS FP16 GEMM kernel, upon which the NestedFP FP16 kernel is built, employs a two-stage pipeline to overlap data transfer and computation. Specifically, the two stages correspond to (1) data movement between shared memory and registers and (2) Tensor Core execution, which are overlapped for high throughput. NestedFP extends this design into a three-stage pipeline to further hide reconstruction overheads. In particular, the three stages in NestedFP are: (1) shared-memory-to-register data transfer, (2) FP16 reconstruction, and (3) Tensor Core execution.

5 Evaluation

Through extensive experiments, we demonstrate that NestedFP is an effective solution for handling fluctuating loads in LLM serving by dynamically switching between FP16 and FP8 modes, as evidenced by the following three key results:

- The quality of FP8 model extracted from FP16 parameters in the NestedFP data format matches state-of-the-art FP8 quantization results. (Section 5.1)
- The FP16 mode of NestedFP achieves throughput comparable to baseline FP16 inference using the vanilla CUTLASS kernel, despite performing on-the-fly FP16 reconstruction. (Section 5.2 and Section 5.3)
- NestedFP significantly improves SLO attainment over FP16-only deployment, without introducing additional memory overhead. (Section 5.4)

5.1 FP8 Model Quality

Methodology. We evaluate the accuracy of the FP8 model in NestedFP against FP8 models quantized using commonly adopted configurations, specifically per-channel weight quantization. For activation quantization, we apply per-token quantization for both NestedFP and the baseline. The evaluation covers six models: Llama 3.1 8B, Mistral Nemo (12B), Phi-4 (14B), Mistral Small (24B), Llama 3.1 70B, and DeepSeek-R1-Distill-Llama-70B [3, 4, 5, 12, 23]. For benchmarks, we use three downstream tasks: Minerva Math, MMLU Pro, and BBH [19, 32, 34]. All models are integrated into the vLLM framework, and accuracy is measured using the LM Evaluation Harness [11]. As noted in Section 4.1, 8.75% of the linear layers in Phi-4 are not applicable to NestedFP and are therefore executed in FP16. More details on the experimental setup are provided in Appendix A.

Results. Table 2 presents the results. Across all models and benchmarks, the FP8 model in NestedFP achieves accuracy comparable to the baseline FP8 model, demonstrating the effectiveness of the new data format of NestedFP.

	Llama 3.1 8B		Mistral Nemo		Phi-4		Mistral Small		Llama 3.1 70B		DeepSeek-R1-Distill-Llama-70B	
	FP8(B)	FP8(N)	FP8(B)	FP8(N)	FP8(B)	FP8(N)	FP8(B)	FP8(N)	FP8(B)	FP8(N)	FP8(B)	FP8(N)
Minerva Math	17.6	17.2	16.5	16.4	42.9	42.9	34.2	35.2	35.7	34.6	46.0	45.5
MMLU Pro	32.8	33.2	35.3	34.7	52.7	53.3	53.6	53.7	48.1	47.4	49.2	49.4
BBH	38.6	39.0	40.6	40.0	27.1	28.7	50.8	51.1	49.1	49.7	53.2	53.4

Table 2: Accuracy of FP8 models in NestedFP across various downstream tasks, compared to the baseline FP8 model. FP8(B) denotes the baseline model, and FP8(N) denotes NestedFP.

5.2 FP16 GEMM Kernel Performance

Methodology. We evaluate the FP16 GEMM kernel performance of NestedFP against the CUTLASS FP16 GEMM kernel, upon which ours is built, using an NVIDIA H100 GPU. Note that NestedFP does not require any custom kernel for FP8 GEMM; thus, we omit FP8 kernel evaluation. We consider four different weight matrix shapes: `(N, K) = (28672, 4096), (28672, 5120), (35840, 5120), (65536, 5120)`. The (M) dimension, which corresponds to the number of tokens processed in parallel, is varied from 32 to 2048 in increments of 32. For both kernels, we select the best configuration (e.g., tile size) for each GEMM shape through an exhaustive search over the design space. Details of the kernel search space are provided in Appendix D.2.

Results. Figure 6 shows the results. The NestedFP kernel consistently demonstrates performance comparable to the CUTLASS baseline across all GEMM shapes. Specifically, the average relative overheads for each weight matrix dimension are 5.89%, 6.33%, 4.59%, and 6.33%, respectively. These results indicate that the NestedFP kernel effectively minimizes the overhead introduced by on-the-fly FP16 reconstruction. Additional results for other GEMM shapes, which exhibit similar trends, as well as comparisons with cuBLAS FP16 kernels, are provided in Appendix B and Appendix D.1.

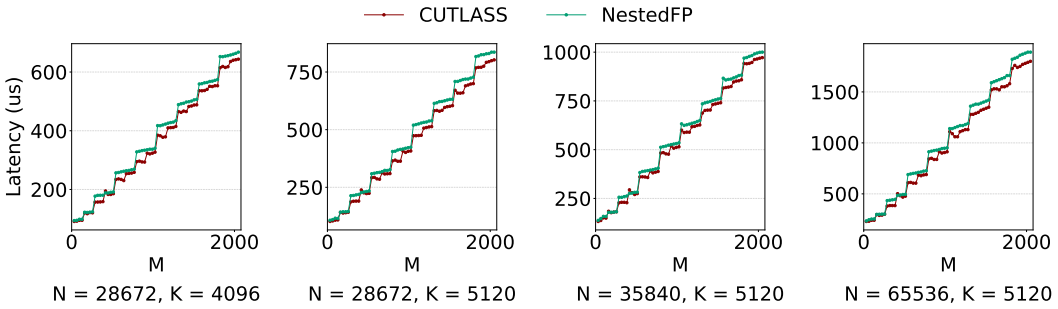


Figure 6: CUTLASS baseline and NestedFP kernel performance comparison.

5.3 FP16 Mode End-to-End Inference Performance

Methodology. We evaluate the end-to-end inference throughput of the FP16 mode of NestedFP by integrating it into the vLLM framework on a single NVIDIA H100 GPU. Specifically, we set the input and output sequence lengths to 1024 and 512, respectively, while varying the batch size from 32 to 512. More details on the experimental setup are provided in Appendix C. The results are compared against vanilla vLLM FP16 execution, which relies on PyTorch for invoking GEMM kernels. Again, the FP8-mode throughput of NestedFP is identical to baselines such as vanilla vLLM and is therefore omitted from evaluation.

Results. Figure 7 presents the results. NestedFP achieves FP16-mode throughput highly comparable to the baseline, with only minor overheads—5.04% (LLaMA 3.1–8B), 5.56% (Mistral Nemo), 4.76% (Phi-4), and 4.56% (Mistral Small) on average across batch sizes. The end-to-end throughput degradation is even lower than the kernel-level overheads reported in Section 5.2, since non-GEMM components also contribute to total execution time. Additional end-to-end throughput results are provided in Appendix C.

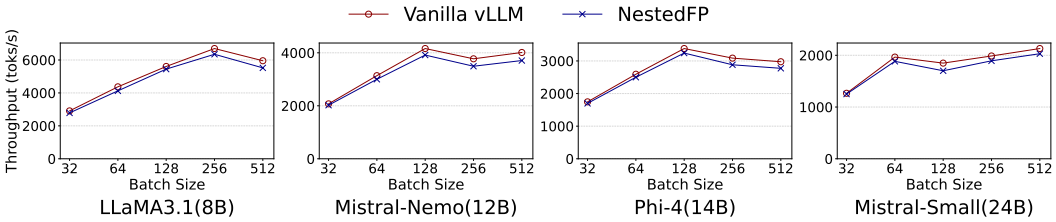


Figure 7: FP16 inference throughput comparison of four models between vanilla vLLM and NestedFP.

5.4 Impact on SLO Attainment

Methodology. In this section, we present a case study demonstrating how NestedFP can improve SLO attainment. We implement a simple load-aware mode switching policy for NestedFP that, in each iteration, selects FP8 mode when the number of tokens to process exceeds a predefined threshold (1024 in this experiment); otherwise, it selects FP16 mode. The evaluation is conducted using Llama 3.1–70B on a cluster equipped with four NVIDIA H100 GPUs. As input, we use 1,000 requests sampled from an Azure LLM inference trace while varying the request rate. We measure p90 TTFT, p90 TPOT, and SLO attainment under two criteria: loose and tight. The loose SLO is defined as $\leq 10\times$ the average single-request latency, and the tight SLO is defined as $\leq 5\times$ the average single-request latency, following the methodology of a previous study [14].

Results. Table 3 shows the results. NestedFP consistently outperforms the FP16-only baseline across all request rates in terms of TTFT, TPOT, and SLO attainment, with particularly pronounced improvements under high request rates. Note that this experiment employs a simple, straightforward switching policy, leaving substantial room for further optimization by integrating NestedFP with more sophisticated policies—such as those considering KV-cache utilization or incorporating load prediction mechanisms. Nevertheless, these results confirm that NestedFP provides an efficient dual-precision mechanism that can significantly enhance SLO attainment in LLM serving systems.

Request Rate	Method	Tight SLO (%)	Loose SLO (%)	p90 TTFT (s)	p90 TPOT (ms)
7.47 req/s	FP16	0.7	14.1	9.9893	162.2
	NestedFP	72.4	90.0	1.9518	118.9
5.03 req/s	FP16	71.6	93.6	1.6449	151.3
	NestedFP	94.6	99.9	0.9679	86.8
3.79 req/s	FP16	83.1	96.9	1.2404	109.2
	NestedFP	96.6	100	0.8168	60.1
1.99 req/s	FP16	90.4	99.4	0.8915	64.9
	NestedFP	98.4	100	0.6116	47.7

Table 3: SLO attainment and p90 TTFT/TPOT of Llama-3.1-70B on 4×H100 under varying loads.

6 Conclusion

We introduce NestedFP, a memory-efficient framework that supports both FP16 and FP8 precisions for LLM inference in a memory-efficient way. NestedFP introduces a new data format that enables direct extraction of FP8 weights from FP16 weights without additional memory overhead. NestedFP also incorporates a custom GEMM kernel optimized for this format to ensure efficient computation. As a result, NestedFP enables dynamic switching between FP16 and FP8 modes with the memory footprint of FP16, without noticeable degradation in either accuracy or inference throughput, providing an effective means to improve SLO attainment in LLM serving.

7 Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MSIT) (RS-2024-00340008) and Institute of Information & Communications Technology Planning & Evaluation (IITP) under the artificial intelligence semiconductor support program (IITP-2023-RS-2023-00256081), funded by the Korea Government (MSIT).

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming Throughput-Latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 117–134, Santa Clara, CA, July 2024. USENIX Association.

[3] DeepSeek AI. Deepseek-r1-distill-llama-70b. `https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Llama-70B`.

[4] Mistral AI. Mistral nemo. `https://huggingface.co/mistralai/Mistral-Nemo-Base-2407`.

[5] Mistral AI. Mistral small. `https://huggingface.co/mistralai/Mistral-Small-24B-Base-2501`.

[6] Microsoft Azure. Azure llm inference trace 2024. `https://github.com/Azure/AzurePublicDataset/blob/master/AzureLLMInferenceDataset2024.md`.

[7] Baseten. Understanding performance benchmarks for llm inference. `https://www.baseten.co/blog/understanding-performance-benchmarks-for-llm-inference`.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[9] Marta Catillo, Umberto Villano, and Massimiliano Rak. A survey on auto-scaling: how to exploit cloud elasticity. *Int. J. Grid Util. Comput.*, 14(1):37–50, January 2023.

[10] Javad Dogani, Reza Namvar, and Farshad Khunjush. Auto-scaling techniques in container-based cloud and edge/fog computing: Taxonomy and survey. *Comput. Commun.*, 209(C):120–150, September 2023.

[11] EleutherAI. lm-evaluation-harness. `https://github.com/EleutherAI/lm-evaluation-harness`.

[12] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[13] Andrew Griffin. Chatgpt creators openai are generating 100 billion words per day, ceo says. `https://www.independent.co.uk/tech/chatgpt-openai-words-sam-altman-b2494900.html`.

[14] Ke Hong, Xiuhong Li, Lufang Chen, Qiuli Mao, Guohao Dai, Xuefei Ning, Shengen Yan, Yun Liang, and Yu Wang. SOLA: Optimizing SLO attainment for large language model serving with state-aware scheduling. In *Eighth Conference on Machine Learning and Systems*, 2025.

[15] Jiwoo Kim, Joonhyung Lee, Gunho Park, Byeongwook Kim, Se Jung Kwon, Dongsoo Lee, and Youngjoo Lee. An investigation of fp8 across accelerators for llm inference. *arXiv preprint arXiv:2502.01070*, 2025.

[16] Andrey Kuzmin, Mart Van Baalen, Yuwei Ren, Markus Nagel, Jorn Peters, and Tijmen Blankevoort. Fp8 quantization: The power of the exponent. *Advances in Neural Information Processing Systems*, 35:14651–14662, 2022.

[17] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[18] Marina Lammertyn. 60+ chatgpt facts and statistics you need to know in 2025. `https://blog.invgate.com/chatgpt-statistics`.

[19] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857, 2022.

[20] Jiachen Liu, Jae-Won Chung, Zhiyu Wu, Fan Lai, Myungjin Lee, and Mosharaf Chowdhury. Andes: Defining and enhancing quality-of-experience in llm-based text streaming services. *arXiv preprint arXiv:2404.16283*, 2024.

[21] Shih-yang Liu, Zechun Liu, Xijie Huang, Pingcheng Dong, and Kwang-Ting Cheng. Llm-fp4: 4-bit floating-point quantized transformers. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, page 592–605. Association for Computational Linguistics, 2023.

[22] Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, et al. Fp8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2022.

[23] Microsoft. Phi-4. `https://huggingface.co/microsoft/phi-4`.

[24] NVIDIA. Cutlass. `https://github.com/NVIDIA/cutlass`.

[25] Muhammad Osama, Duane Merrill, Cris Cecka, Michael Garland, and John D Owens. Stream-k: Work-centric parallel decomposition for dense matrix-matrix multiplication on the gpu. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, pages 429–431, 2023.

[26] Yeonhong Park, Jake Hyun, SangLyul Cho, Bonggeun Sim, and Jae W. Lee. Any-precision llm: Low-cost deployment of multiple, different-sized llms. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

[27] Sergio P Perez, Yan Zhang, James Briggs, Charlie Blake, Josh Levy-Kramer, Paul Balanca, Carlo Luschi, Stephen Barlow, and Andrew William Fitzgibbon. Training and inference of large language models using 8-bit floating point. *arXiv preprint arXiv:2309.17224*, 2023.

[28] Haihao Shen, Naveen Mellempudi, Xin He, Qun Gao, Chang Wang, and Mengni Wang. Efficient post-training quantization with fp8 formats. *Proceedings of Machine Learning and Systems*, 6:483–498, 2024.

[29] Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. Llumnix: Dynamic scheduling for large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 173–191, 2024.

[30] Ting Sun, Penghan Wang, and Fan Lai. Hygen: Efficient llm serving via elastic online-offline request co-location. *arXiv preprint arXiv:2501.14808*, 2025.

[31] Xiao Sun, Jungwook Choi, Chia-Yu Chen, Naigang Wang, Swagath Venkataramani, Vijayalakshmi Viji Srinivasan, Xiaodong Cui, Wei Zhang, and Kailash Gopalakrishnan. Hybrid 8-bit floating point (hfp8) training and inference for deep neural networks. *Advances in neural information processing systems*, 32, 2019.

[32] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

[33] vllm project. vllm. `https://github.com/vllm-project/vllm`.

[34] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

[35] Haojun Xia, Zhen Zheng, Xiaoxia Wu, Shiyang Chen, Zhewei Yao, Stephen Youn, Arash Bakhtiari, Michael Wyatt, Donglin Zhuang, Zhongzhu Zhou, et al. {Quant-LLM}: Accelerating the serving of large language models via {FP6-Centric}{Algorithm-System}{Co-Design} on modern {GPUs}. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 699–713, 2024.

[36] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[37] Yuhang Yao, Han Jin, Alay Dilipbhai Shah, Shanshan Han, Zijian Hu, Yide Ran, Dimitris Stripelis, Zhaozhuo Xu, Salman Avestimehr, and Chaoyang He. Scalellm: A resource-frugal llm serving framework by optimizing end-to-end efficiency. *arXiv preprint arXiv:2408.00008*, 2024.

[38] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, 2022.

[39] Tianyi Zhang, Yang Sui, Shaochen Zhong, Vipin Chaudhary, Xia Hu, and Anshumali Shrivastava. 70% size, 100% accuracy: Lossless llm compression for efficient gpu inference via dynamic-length float. *arXiv preprint arXiv:2504.11651*, 2025.

[40] Yijia Zhang, Lingran Zhao, Shijie Cao, Sicheng Zhang, Wenqiang Wang, Ting Cao, Fan Yang, Mao Yang, Shanghang Zhang, and Ningyi Xu. Integer or floating point? new outlooks for low-bit quantization on large language models. In *2024 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2024.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We have made our contributions visible to abstract and introduction.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: As discussed in the main text, performance differences relative to the baseline may fluctuate depending on the batch size. To ensure transparency, we present performance graphs covering batch sizes in the range [32, 2048], with increments of 32.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: This paper does not contain theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We have disclosed all the design space configurations.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We will provide code along with supplementary materials.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We have disclosed all the design space configurations.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We do not include statistical significance analysis, as it is not the focus of this work.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We have specified that our experiments were conducted using an H100 GPU.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have followed the Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Yes, we have explained LLM inference serving can be facilitated with our mechanism.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: We have no such risk.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Yes, we have cited all the open source projects we are using.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Yes, we will provide instructions to reproduce our experiments and as well code in appendix.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: Our research does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.

- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: Yes, we have used LLM for writing and visualization and the core methods are original.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Evaluation Details

We conducted our evaluation using the Mistral Nemo Base 2407 and Mistral Small 24B Base 2501 models. For accuracy assessment, we employed the Math Verify Metric on the Minerva Math task, the Open LLM Leaderboard configuration (5-shot, multiple choice) for the MMLU Pro task, and a zero-shot evaluation setting for the BBH task. Kernel performance was measured using NVIDIA Nsight Compute. In order to get optimal kernel, we padded activation in the multiple of tile dimension T_m. During performance measurement, we flushed all caches using the cache-control option, but did not fix the GPU clock frequency using the clock-control option.

B Extended Kernel Performance for NestedFP

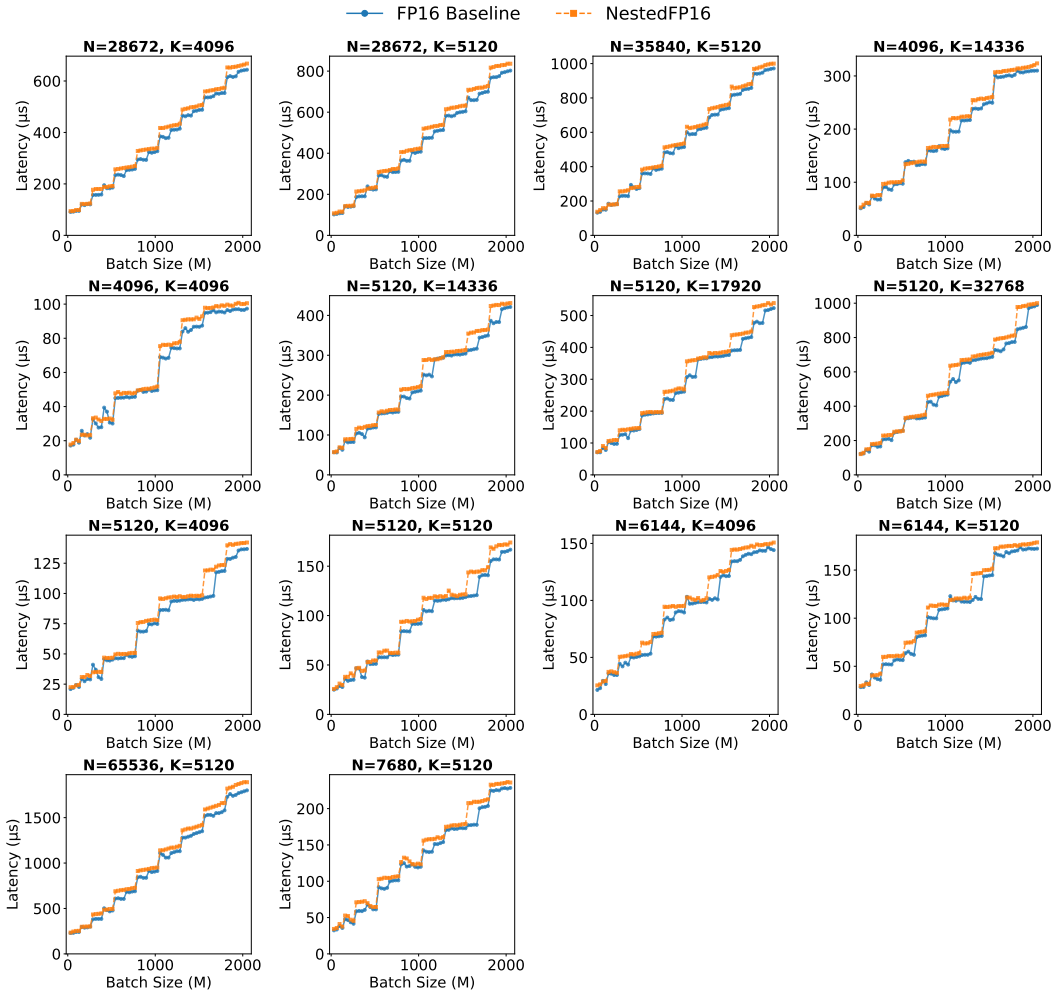
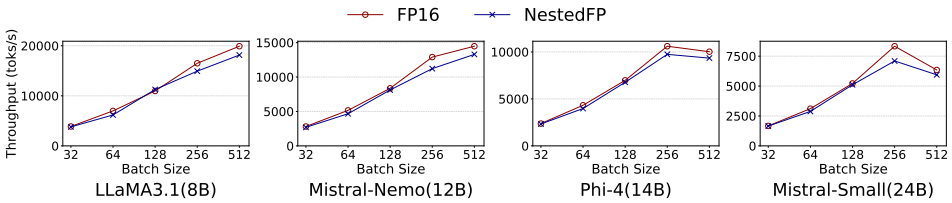


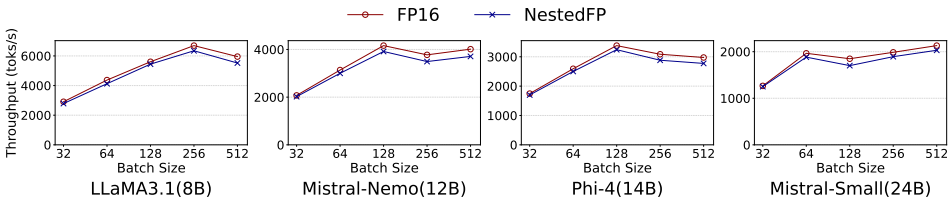
Figure 8: Performance comparison between our CUTLASS baseline and NestedFP kernel across 14 unique (N, K) GEMM shapes. Each subplot represents a different shape configuration, with batch size (M) on the x-axis and latency on the y-axis.

Figure 8 compares our method against the CUTLASS FP16 baseline across all 14 GEMM shapes for the four evaluated models: Llama 3.1 8B, Mistral Nemo, Phi-4, and Mistral Small. The matrix dimension M is varied in increments of 32, ranging from 32 to 2048. Our results show that NestedFP consistently incurs only moderate overhead across all GEMM configurations, with an average performance difference of 6.1%. The per-graph average overhead ranges from 4.3% to 7.2%, demonstrating stable performance regardless of matrix shape. This consistency highlights the effectiveness of our kernel-level optimizations.

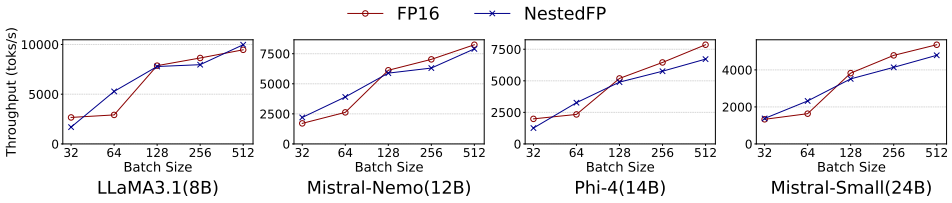
C Extended End-to-End Throughput Evaluation for NestedFP



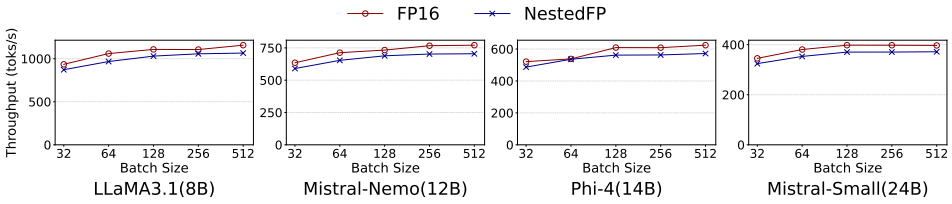
(a) (32, 512)



(b) (1024, 512)



(c) (32, 32)



(d) (1024, 32)

Figure 9: Extended end-to-end throughput evaluation across different input/output token configurations. Each subplot shows throughput comparison for FP16 and NestedFP across four models (Llama 3.1 8B, Mistral Nemo, Phi-4, Mistral Small) under different request size settings: (a) 32 input, 512 output tokens (b) 1024 input, 512 output tokens (c) 32 input, 32 output tokens and (d) 1024 input, 32 output tokens.

We extend our end-to-end evaluation to further demonstrate the robustness of NestedFP. Figure 9 presents results for four additional input/output configurations: (32, 512), (1024, 512), (32, 32), and (1024, 32). Across all settings, NestedFP exhibits minimal overhead compared to the FP16 baseline, with an average degradation of 5.52% and a maximum of 7.69%. On a per-model basis, the average overhead is 5.97% for LLaMA 3.1 (8B), 6.67% for Mistral-Nemo (12B), 5.51% for Phi-4 (14B), and 5.05% for Mistral-Small (24B).

Our evaluation setup mirrors the configuration used in the main throughput experiments. Throughput was computed as the total number of generated tokens divided by the end-to-end latency for each test. All tests were conducted on a single NVIDIA H100 GPU using the vLLM 0.8.5 V1 engine with PyTorch 2.7.0+cu12.6 for both the baseline and our proposed NestedFP implementations. Chunked prefill, enabled by default in the vLLM V1 engine, was used throughout the evaluation. This feature segments the prefill phase into smaller chunks, which are processed concurrently with decoding tokens within the same batch. At each iteration, the GEMM operation shape (M, N, K) is determined based on the number of tokens grouped in that step, with M denoting the total token count. We set the maximum number of batched tokens to 8192 for the throughput test.

D CUTLASS FP16 GEMM Kernel

D.1 CUTLASS FP16 Baseline and PyTorch Kernel Comparison

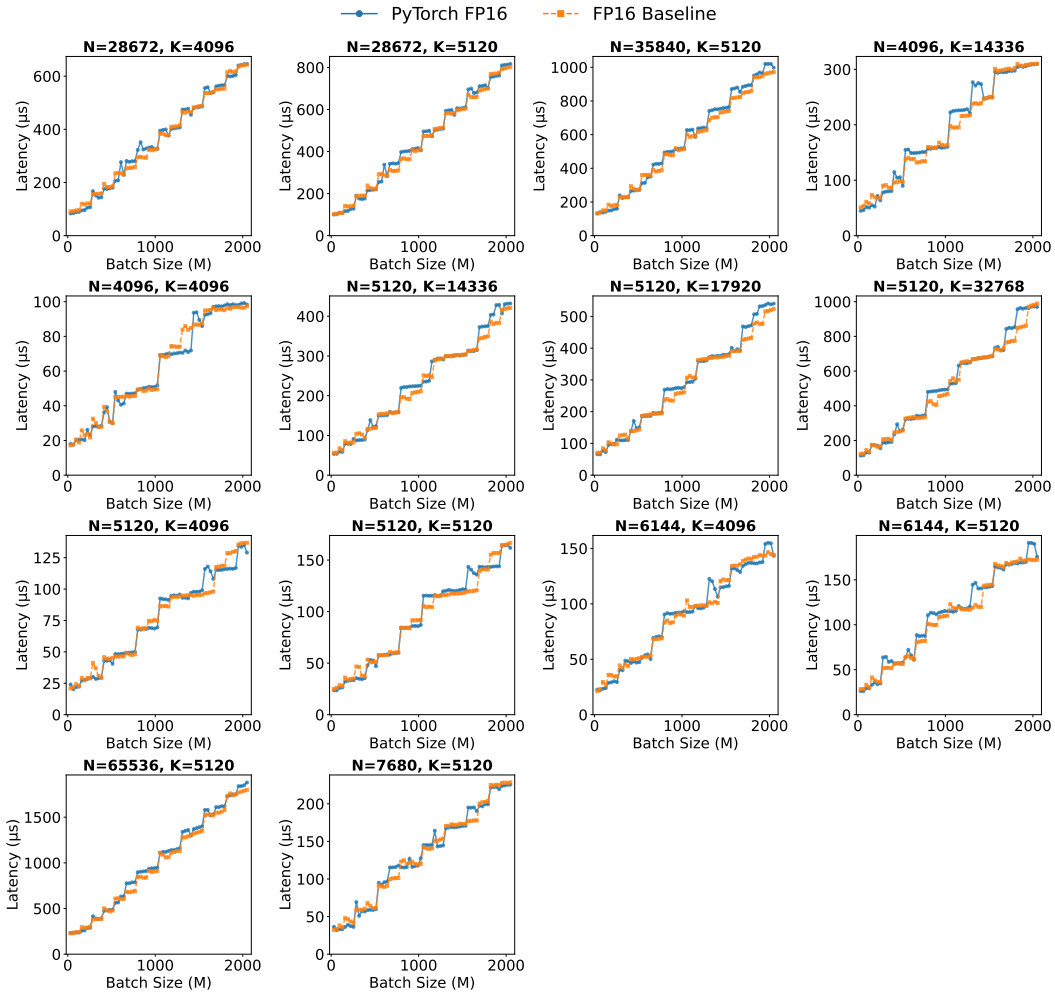


Figure 10: Performance comparison between PyTorch kernel and our CUTLASS baseline across 14 unique (N, K) GEMM shapes. Each subplot represents a different shape configuration, with batch size (M) on the x-axis and latency on the y-axis.

To validate the robustness of our CUTLASS baseline, we compare its performance against the cuBLAS kernels employed by PyTorch 2.6.0 with CUDA 12.4. Using a weak baseline could underestimate the overhead introduced by NestedFP, leading to overly optimistic conclusions about its practical efficiency. Figure 10 provides a detailed comparison across 14 (N, K) configurations, extracted from the linear layers of the four models evaluated in our main experiments—Llama 3.1 8B, Mistral Nemo, Phi-4, and Mistral Small. For each configuration, we sweep the batch size M from 32 to 2048 in increments of 32, covering the typical range encountered in LLM inference workloads.

Our results show that the FP16 baseline closely matches the performance of the cuBLAS implementation, with an average performance difference of just 1.8% across all 14 GEMM shapes. In 11 of the 14 configurations, our baseline achieves higher average performance than PyTorch, with performance differences reaching up to 3.3%. For the remaining three shapes—corresponding to smaller GEMMs—PyTorch slightly outperforms our implementation, with differences ranging from 0.3% to 0.9%, well within the bounds of normal GPU performance variability. These results confirm that our CUTLASS baseline is not only competitive but often exceeds production-level kernel performance, ensuring a reliable and fair comparison when evaluating NestedFP's overhead.

D.2 Kernel Search Space

The kernel design space consists of two variants: non-cooperative and cooperative kernels. For non-cooperative kernels, we perform a grid search over tile dimensions with $T_m \in 16, 32, 64, 128, 256$, $T_n \in 64, 128, 256$, and $T_k \in 64, 128, 256$. Cooperative kernels adopt larger tiles with $T_n \in 128, 256$, and configurations that fail to compile are excluded. Non-cooperative kernels use a cluster shape of $(1, 1, 1)$, whereas cooperative kernels use $(2, 1, 1)$. We employ the persistent scheduler for non-cooperative kernels, and both the persistent and Stream-K [25] schedulers for cooperative kernels.

E Broad Applicability of NestedFP

Model	GEMM1	GEMM2	GEMM3	GEMM4	Total
CodeLlama 7B	96/96	32/32	64/64	31/32	223/224 (99.6%)
CodeLlama 13B	120/120	40/40	80/80	37/40	277/280 (98.9%)
Gemma 3 4B	207/264	64/88	123/176	34/34	429/563 (76.2%)
Gemma 3 12B	249/306	78/102	151/204	48/48	527/661 (79.7%)
Gemma 3 27B	291/348	92/116	179/232	62/62	625/759 (82.3%)
Llama 3.1 8B	96/96	32/32	64/64	32/32	224/224 (100.0%)
Llama 3.1 70B	224/240	80/80	141/160	78/80	523/560 (93.4%)
Mistral Nemo 12B	120/120	40/40	80/80	40/40	280/280 (100.0%)
Mistral Small 24B	120/120	40/40	80/80	40/40	280/280 (100.0%)
Phi-3.5 Mini	26/32	31/32	31/32	24/32	112/128 (87.5%)
Phi-4 14B	40/40	38/40	40/40	28/40	146/160 (91.2%)
Qwen 3 8B	108/108	35/36	72/72	34/36	249/252 (98.8%)
Qwen 3 14B	120/120	40/40	80/80	38/40	278/280 (99.3%)
Qwen 3 32B	192/192	63/64	127/128	56/64	438/448 (97.8%)

Table 4: Layer-wise applicability of NestedFP across models. Format X/Y indicates X applicable layers out of Y total layers for each GEMM type.

Our scheme is broadly applicable to a wide range of LLMs beyond those evaluated in the main experiments. In Table 4, we analyze ten additional models that exhibit the key characteristic: the majority of their weight values have absolute magnitudes less than or equal to 1.75. For each model, we examine four distinct GEMM shapes corresponding to different linear layer types: GEMM1 (QKV projections), GEMM2 (output projections), GEMM3 (MLP gate/up projections), and GEMM4 (MLP down projections). Each entry is shown as X/Y, indicating that X out of Y layers of that type satisfy the weight magnitude constraint (absolute value ≤ 1.75).

Our analysis indicates that most models exhibit high applicability rates. The Gemma 3 family shows the lowest rates (76.2%–82.3%), primarily due to multi-modal projection layers containing weights with significantly higher magnitudes (up to 26.25). Among the various GEMM types, GEMM4 (MLP down projections) tends to include more layers exceeding the threshold in models such as Phi-4 14B (70.0% applicability) and Qwen 3 32B (87.5%). While the majority of models have maximum weight values below 3.0, notable outliers include Llama 3.1 70B (maximum value of 93.0) and the Gemma 3 series (maximum value of 26.25). These extreme values are typically confined to a small number of layers and have limited impact on the overall applicability of our method. The consistently high applicability rates across diverse model families affirm the robustness and generalizability of our approach's core assumption regarding weight distributions.