

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 as it provides more robust performance. 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 NestedFP16

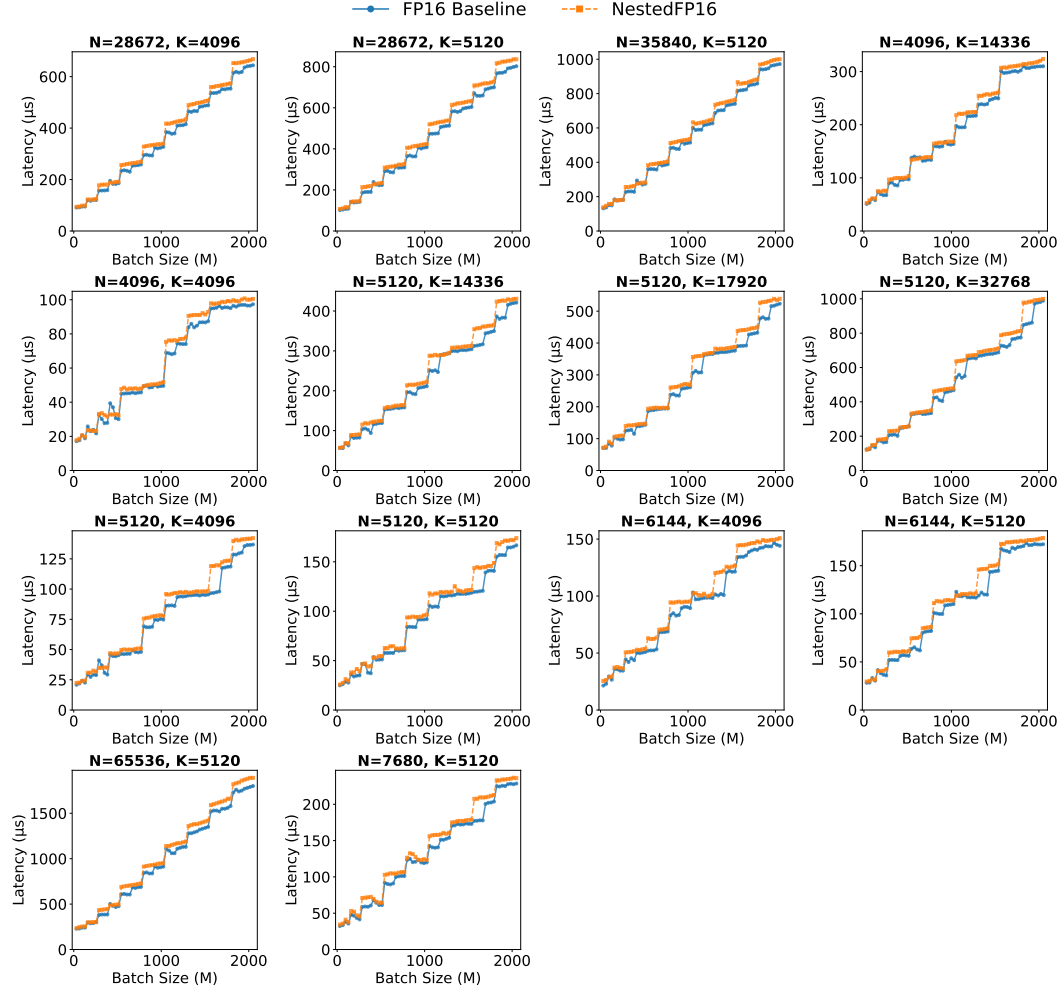


Figure 1: Performance comparison between our CUTLASS baseline and NestedFP16 kernel across 14 unique (N, K) GEMM shapes. Each subplot represents a different shape configuration, with batch size (M) on the x -axis and latency on the y -axis.

Figure 1 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

16 7.2%, demonstrating stable performance regardless of matrix shape. This consistency highlights the
 17 effectiveness of our kernel-level optimizations.

18 C Extended End-to-End Throughput Evaluation

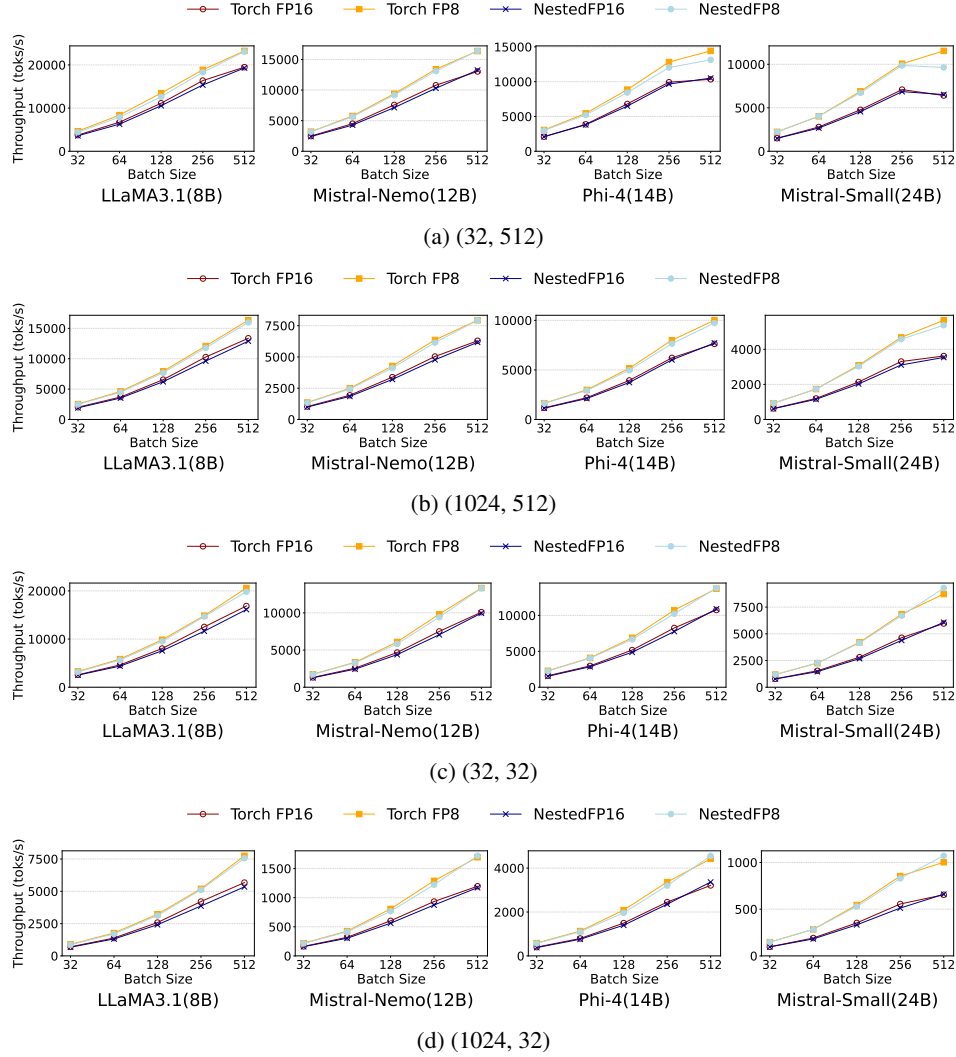


Figure 2: Extended end-to-end throughput evaluation across different input/output token configurations. Each subplot shows throughput comparison for FP16, NestedFP16, FP8 and NestedFP8 across four models (LLama 3.1 8B, Mistral Nemo, Phi-4, Mistral Small) under different request size settings: (a) 32 input, 512 output tokens (b) 1024 input, 512 output tokens (c) 32 input, 32 output tokens and (d) 1024 input, 32 output tokens.

19 We extend our end-to-end evaluation to further demonstrate the robustness of NestedFP. Figure 2
 20 presents results for four additional input/output configurations: (32, 512), (1024, 512), (32, 32),
 21 and (1024, 32). Across all settings, NestedFP16 exhibits minimal overhead compared to the FP16
 22 baseline, with an average degradation of 3.88% and a maximum of 8.22%. On a per-model basis,
 23 the average overhead is 5.15% for LLaMA 3.1 (8B), 4.19% for Mistral-Nemo (12B), 3.35% for
 24 Mistral-Small (24B), and 2.81% for Phi-4 (14B). In contrast, NestedFP8 achieves significant speedups
 25 over NestedFP16, with average throughput gains of $1.26\times$ for LLaMA 3.1, $1.33\times$ for Mistral-Nemo,
 26 $1.53\times$ for Mistral-Small, and $1.36\times$ for Phi-4 across all input/output shapes. Notably, the largest
 27 model, Mistral-Small, benefits the most from the NestedFP8 quantization. These results indicate that
 28 the performance gains of NestedFP are consistent and robust across a wide range of input/output
 29 configurations.

When compared to Torch FP8, NestedFP8 maintains competitive performance, achieving 96.84% of Torch FP8’s throughput for LLaMA 3.1, 97.76% for Mistral-Nemo, 98.79% for Mistral-Small, and 96.77% for Phi-4. Our limited search space shows potential gain that can be achieved with further FP8 kernel optimization. We note that we did adopt Stream-K scheduling for NestedFP8.

Our evaluation setup mirrors the configuration used in the main throughput experiments. All tests were conducted on a single NVIDIA H100 GPU using the vLLM 0.8.3 V1 engine with PyTorch 2.6.0+cu12.4 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. To systematically evaluate throughput under different batch sizes, we adjusted the `max_num_batched_token` and `max_num_seqs` parameters to constrain the batch size accordingly.

D CUTLASS FP16 GEMM Kernel

We extend the open-source NVIDIA CUTLASS 3.6 library to support our custom format. We begin by describing the efficient design of GEMM kernels within CUTLASS for the NVIDIA H100 GPU and introduce key components that define our design space. In the following section, we benchmark our CUTLASS kernels against PyTorch to validate strong performance of CUTLASS baseline.

D.1 GEMM Kernel Design

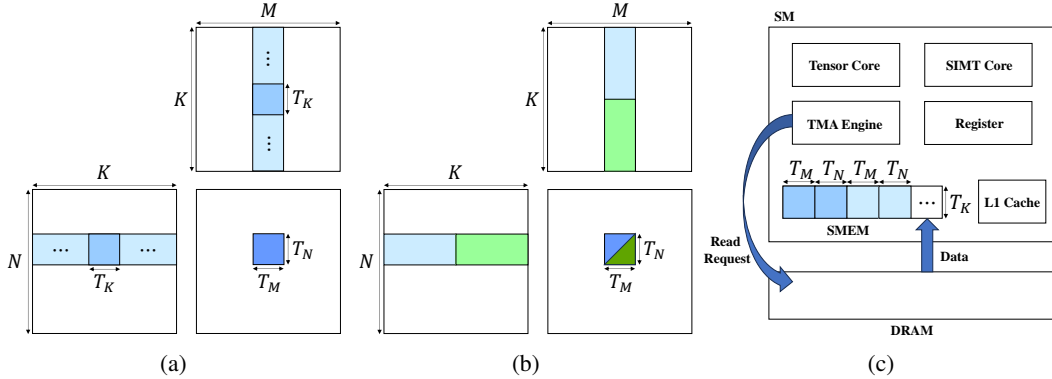


Figure 3: (a) Data parallel scheduling. A single output tile is processed by a single thread block. (b) Stream-K scheduling. A single output tile is processed collaboratively across multiple thread blocks. (c) DRAM to shared memory copy handled by 4 warps in producer warp group and TMA engine.

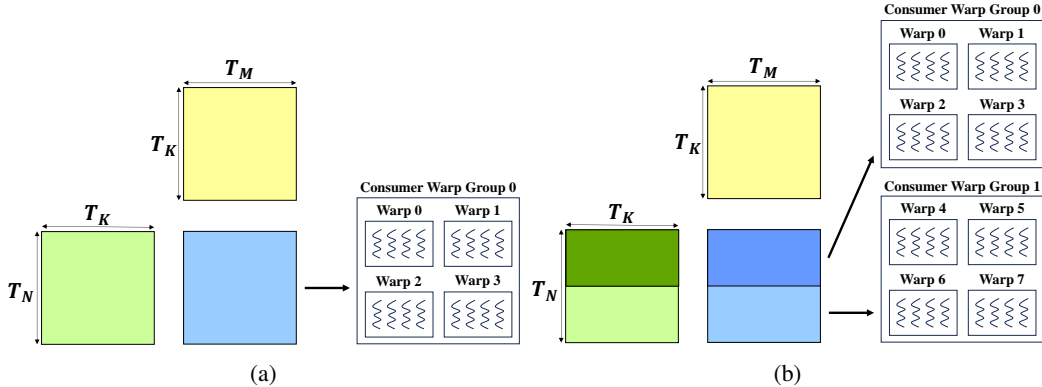


Figure 4: (a) Non-cooperative kernel. A single warp group is responsible for processing entire input tile. (b) Cooperative kernel. Two warp groups are each responsible for processing half of the first input tile and corresponding output tile.

49 GEMM operates on a weight matrix of shape $N \times K$ and an activation matrix of shape $M \times K$. To
 50 parallelize this computation, CUTLASS partitions the input matrices into tiles of size $T_n \times T_k$ and
 51 $T_m \times T_k$, respectively. These tiles define the granularity at which work is distributed across thread
 52 blocks.

53 As illustrated in Figure 3a, each input tile is processed along the K dimension to produce a corre-
 54 sponding output tile, which is accumulated in registers. CUTLASS employs a persistent tile scheduler,
 55 assigning each thread block to a fixed streaming multiprocessor (SM) for the duration of execution.
 56 In **data-parallel scheduling**, each SM processes the full K dimension for a given output tile indepen-
 57 dently. In contrast, **stream-K scheduling** enables multiple thread blocks to collaboratively compute a
 58 single output tile by dividing the K -dimension workload across thread blocks, as shown in Figure 3b.
 59 The partial results are later merged to form the final output. In all cases, the computed output tiles are
 60 written back to global memory (DRAM).

61 To hide DRAM latency and enable compute-memory overlap, tiles are **asynchronously** loaded
 62 from global memory into shared memory. As shown in Figure 3c, shared memory is partitioned to
 63 hold multiple tiles, allowing future iterations to be staged while the current tile is being processed.
 64 On NVIDIA H100 GPUs, this transfer is managed by the **Tensor Memory Accelerator (TMA)**
 65 engine. CUTLASS uses warp specialization to pipeline these operations: one warp group (four warps)
 66 handles TMA data transfers (producers), while one or two other warp groups (consumers) execute
 67 matrix-multiply-accumulate (MMA) operations, depending on the kernel configuration.

68 Once data is available in shared memory, the kernel proceeds with MMA using one of two config-
 69 urations. In the RS (register-shared) kernel, weights are loaded into registers and processed with
 70 activations from shared memory—this approach enables bit-level manipulations required by our
 71 custom format. In contrast, SS (shared-shared) kernels source both operands directly from shared
 72 memory. On Hopper, SM90 MMA instructions require the second operand to come from shared
 73 memory. **MMA operations** are executed at the **warp group granularity**, requiring all warps in a
 74 group to be ready before issuing the instruction. Consequently, SIMT-level manipulations on weights
 75 must complete across all four warps before any warp group MMA operation begins.

76 CUTLASS supports both **non-cooperative** and **cooperative** kernel modes. Non-cooperative kernels
 77 use a single consumer warp group, while cooperative kernels employ two consumer groups that share
 78 activations but process distinct weight tiles. The behavioral differences between these modes are
 79 illustrated in Figure 4.

80 D.2 CUTLASS FP16 Baseline and cuBLAS W16A16 Kernel Comparison

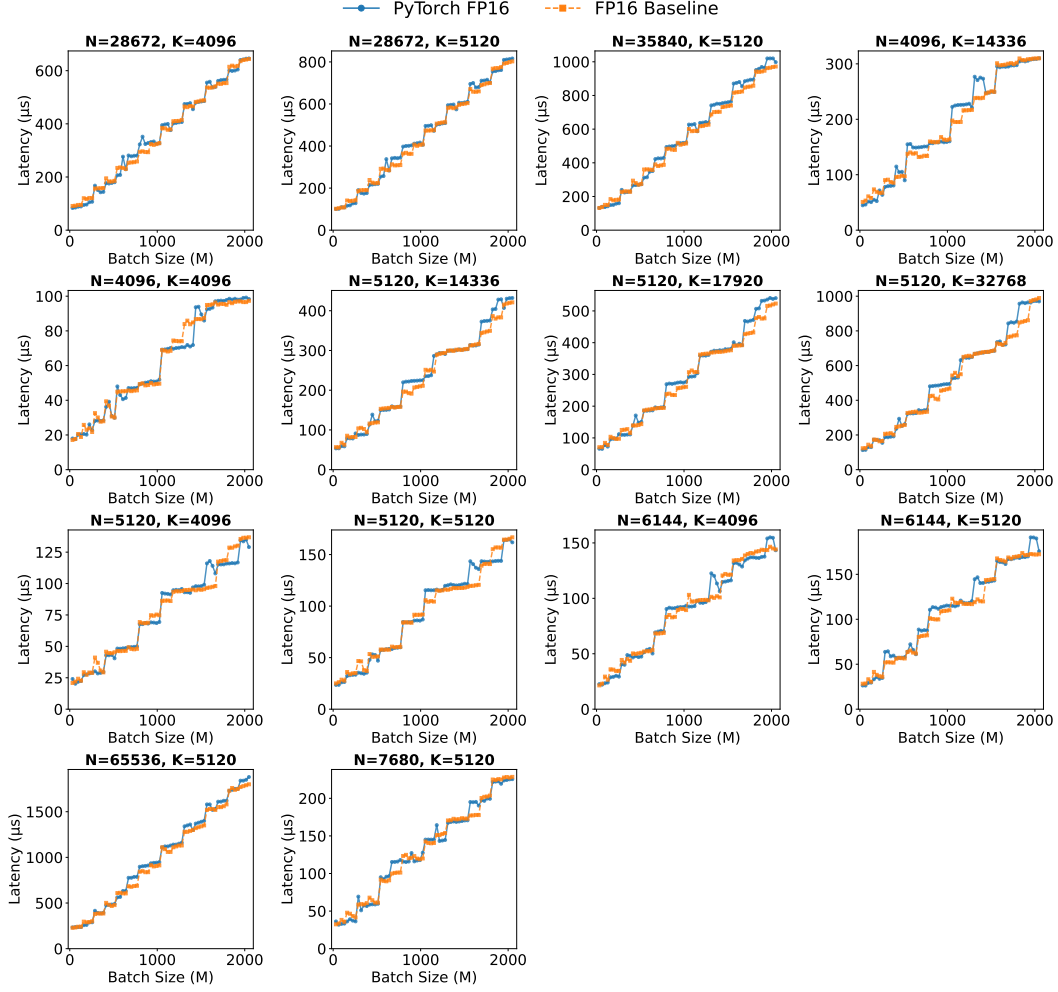


Figure 5: Performance comparison between PyTorch `torch.matmul` and our CUTLASS baseline across 14 unique (N, K) GEMM shapes. Each subplot represents a different shape configuration, with batch size (M) on the x -axis and latency on the y -axis.

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 5 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.

96 **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 1: Layer-wise applicability of NestedFP across models. Format X/Y indicates X applicable layers out of Y total layers for each GEMM type.

97 Our scheme is broadly applicable to a wide range of LLMs beyond those evaluated in the main
 98 experiments. In Table 1, we analyze ten additional models that exhibit the key characteristic: the
 99 majority of their weight values have absolute magnitudes less than or equal to 1.75. For each model,
 100 we examine four distinct GEMM shapes corresponding to different linear layer types: GEMM1 (QKV
 101 projections), GEMM2 (output projections), GEMM3 (MLP gate/up projections), and GEMM4 (MLP
 102 down projections). Each entry is shown as X/Y, indicating that X out of Y layers of that type satisfy
 103 the weight magnitude constraint (absolute value ≤ 1.75).

104 Our analysis indicates that most models exhibit high applicability rates. The Gemma 3 family shows
 105 the lowest rates (76.2%–82.3%), primarily due to multi-modal projection layers containing weights
 106 with significantly higher magnitudes (up to 26.25). Among the various GEMM types, GEMM4 (MLP
 107 down projections) tends to include more layers exceeding the threshold in models such as Phi-4 14B
 108 (70.0% applicability) and Qwen 3 32B (87.5%). While the majority of models have maximum weight
 109 values below 3.0, notable outliers include Llama 3.1 70B (maximum value of 93.0) and the Gemma 3
 110 series (maximum value of 26.25). These extreme values are typically confined to a small number
 111 of layers and have limited impact on the overall applicability of our method. The consistently high
 112 applicability rates across diverse model families affirm the robustness and generalizability of our
 113 approach’s core assumption regarding weight distributions.