Why Smaller Is Slower? Dimensional Misalignment in Compressed LLMs

Jihao Xin KAUST
Tian Lyu KAUST
Qilong Pan HUMAIN
Kesen Wang HUMAIN
Marco Canini KAUST

Abstract

Post-training compression reduces LLM parameter counts but often produces irregular tensor dimensions that degrade GPU performance—a phenomenon we call *dimensional misalignment*. We present a full-stack analysis tracing root causes at three levels: framework, library, and hardware. The key insight is that model inference becomes slower because the resulting dimensions are unfriendly with the GPU execution stack. For example, compressing Llama-3-8B with activation-aware singular value decomposition (ASVD) has 15% fewer parameters yet runs no faster than the uncompressed baseline, because 95% of its dimensions are misaligned.

We propose **GAC** (GPU-Aligned Compression), a new compression paradigm that wraps any dimension-reducing compressor and re-selects hardware-aligned dimensions via multi-choice knapsack optimization under the same parameter budget. We evaluate GAC on Llama-3-8B with ASVD and LLM-Pruner, achieving 100% alignment and recovering up to 1.5× speedup while preserving model quality.

CCS Concepts: • **Computing methodologies** → **Neural networks**.

ACM Reference Format:
Jihao Xin, Tian Lyu, Qilong Pan, Kesen Wang, and Marco Canini. 2026. Why Smaller Is Slower? Dimensional Misalignment in Compressed LLMs. In *The 6th Workshop on Machine Learning and Systems (EuroMLSys '26), April 27–30, 2026, Edinburgh, Scotland Uk.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3805621.3807608

1 Introduction

Large Language Models (LLMs) deliver strong capabilities, but their scale hinders deployment. Post-training compression reduces model size and can be categorized into three families: *quantization* [5, 7], *sparsification* [4, 16], and *dimension reduction* [1, 2, 20]. Quantization and sparsification preserve tensor shapes and are therefore compatible with existing GPU kernels. Pretrained models come with carefully designed dimensions that are GPU-friendly, such as 8-aligned dimensions for FlashAttention [3]. Dimension reduction, however, alters the tensor shapes, often producing *irregular* dimensions (e.g., dimension reduced from 128 to 107), which conflict with GPU execution primitives—Tensor Core tiles, cuBLAS kernels, and framework backend selection—causing a counterintuitive outcome: *models with **fewer** parameters run **slower** than uncompressed ones*. We term this phenomenon **dimensional misalignment**.

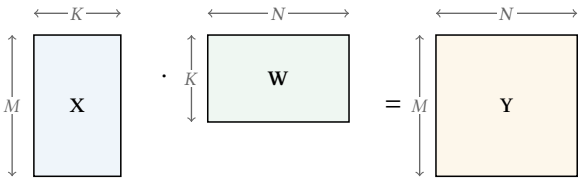


Figure 1. GEMM: $Y{=}X{\cdot}W$ with $X{\in}\mathbb{R}^{M\times K}$, $W{\in}\mathbb{R}^{K\times N}$.

To see why, consider one of the core operators in LLMs, GEMM (General Matrix Multiply) in the expression: $Y{=}X{\cdot}W$ (Figure 1), with sequence dimension M, inner dimension K, and output dimension N. Three compression methods each target a different axis: **(i)** low-rank factorization replaces W with two factors $A{\cdot}B$, introducing a reduced rank r as the inner dimension of two smaller GEMMs [2, 18, 20]; **(ii)** structured pruning removes neurons or attention heads, shrinking N [8]; **(iii)** token/KV eviction drops sequence entries, reducing M [1, 21]. All three optimize *size vs. accuracy* without checking whether the resulting dimensions satisfy GPU alignment constraints (e.g., whether $d \bmod 8 = 0$). The problem is pervasive: ASVD [20] produces 95% misaligned dimensions, and even LLM-Pruner [8]—whose pruning granularity is relatively coarse—still leaves 17% of weights misaligned (Table 5). Prior hardware-aware methods [15, 19] treat latency as a black-box, tying solutions on specific architectures such as CNNs without isolating root causes or providing parameter-budget guarantees.

We propose **GAC** (GPU-Aligned Compression), a compression *paradigm* that imposes hardware alignment constraints on any dimension-reducing compressor. GAC first identifies *why* certain dimensions are slow via a full-stack analysis (§3); next, it constrains a multi-choice knapsack optimizer to re-select aligned dimensions under the same parameter budget. GAC is a compressor-agnostic framework that restores speed without changing the upstream compressor's design.

Contributions.

- We identify the *dimensional misalignment* problem in compressed LLMs and demonstrate its prevalence across compressors (§2).

EuroMLSys '26, Edinburgh, Scotland Uk

ACM ISBN 979-8-4007-2605-7/26/04
https://doi.org/10.1145/3805621.3807608

- We conduct a systematic full-stack analysis to trace the root causes and categories them into three layers: framework, library, and hardware(§3).
- We formalize the dimension selection as a constrained optimization and provide a dynamic programming solver to guarantee the alignment (§4).
- Preliminary evaluation on Llama-3-8B [9] achieves up to 1.5× speedup without sacrificing accuracy (§5).

2 Motivation: Dimensional Misalignment

Existing LLM compression aims to preserve accuracy under a given compression ratio ρ. Formally, given the set of pre-trained parameters $\mathcal{W}$, we seek a compressed set $\mathcal{W}'$ that minimizes expected loss subject to the size constraint:

$$\min_{\mathcal{W}'} \mathbb{E}_{(x,y)\sim\mathcal{D}}[\mathcal{L}(\mathcal{W}';x,y)] \quad \text{s.t.} \quad 1-|\mathcal{W}'|/|\mathcal{W}| \geq \rho. \quad (1)$$

Here $\mathcal{D}$ is the data distribution, $\mathcal{L}$ is the loss, $|\mathcal{W}'|$ and $|\mathcal{W}|$ denote total parameter counts. We denote $B = (1-\rho)|\mathcal{W}|$ as the parameter budget.

Different parameters have different compression sensitivities, e.g., early and late layers are often more critical than middle layers—so budget cannot be allocated uniformly. Existing methods proceed in two steps. First, for each parameter W_i, compute an **importance score** s_i using a proxy (Table 1); a higher s_i reflects higher sensitivity, so we should retain more parameters in W_i. Second, allocate the budget B by assigning each W_i a dimension d_i so that more important W_i receive larger d_i:

$$\{d_i\} = \arg\max_{\{d_i\}} \sum_i s_i \cdot |W_i| \quad \text{s.t.} \quad |\mathcal{W}'| \leq B, \quad d_i \geq 0. \quad (2)$$

Here d_i is the compressed dimension (e.g., inner dimension or output width, varies by compression method), $|W_i|$ is the parameter count given dimension d_i, $|\mathcal{W}'| = \sum_i |W_i|$ is the total parameter count of the compressed model, and B is the parameter budget. Because s_i and the optimum $\{d_i\}$ are continuous, the resulting dimensions are typically *irregular* (e.g., 107, 108, 109) and often violate GPU alignment requirements (e.g. $d_i \bmod 8 = 0$), triggering backend fallbacks and kernel switches that erase the expected speedup from fewer parameters and FLOPs.

We demonstrate the dimensional misalignment problem with a real-world example: PaLU [2]. We use **8-alignment** ($d \bmod 8 = 0$) as an example of alignment constraints. When dimensions are not multiples of 8, latency can increase by up to 90% (Figure 5); we detail this analysis in §3. Figure 3 shows that a large fraction of layers end up misaligned (e.g., 78% in this setup). For generality, we summarize the mainstream importance score proxies into four categories (Table 1). We empirically measure unconstrained dimension allocation on Llama-3-8B at ρ=20% with all four proxies; Figure 2 demonstrates that misalignment consistently occurs across methods. Appendix A shows the misalignment persists across compression ratios $\rho \in [10\%, 50\%]$.

Table 1. Importance scores for budget allocation.

Method	Score	Works
Magnitude	$\|W_i\|_F$	SVD-LLM [18]
Activation	$\|X_i\|_F$	ASVD [20]
Gradient	$\left\|\frac{\partial \mathcal{L}}{\partial h_i} \cdot h_i\right\|$	Taylor Pruning [10]
Fisher	$\mathrm{tr}(\mathbf{F}_i)$	PaLU [2]

3 Full-Stack Analysis

We analyze where and why dimensional misalignment causes slowdowns on an NVIDIA A100-80GB with PyTorch 2.9.1, CUDA 12.8, FP16. Latency is measured with CUDA events (50 warmup, 200 measurement iterations, 3 trials). Root causes fall into three layers (Figure 4): Framework, Library, and Hardware. We detail each layer in the following subsections.

3.1 Framework Layer

Existing DL frameworks such as PyTorch and TensorFlow dispatch an operation to different backends. For example, a matrix multiply `A@B` in PyTorch may run via cuBLAS or via a Triton kernel depending on the shape and hardware, where the dispatching mechanism is hidden from the user. In this section, we exemplify this issue via PyTorch's **SDPA** (Scaled Dot-Product Attention), the core attention mechanism:

$$\text{Attention}(Q,K,V) = \text{softmax}(QK^\top/\sqrt{d_k})V \quad (3)$$

where Q, K, V are the query, key, and value matrices and d_k is the per-head dimension. When one calls the SDPA API[1], PyTorch may select an optimized implementation (e.g., FlashAttention) or fall back to a naive eager implementation (the "Math" backend).

We measured SDPA latency with inputs Q, K, V of shape (B, S, H, d): batch B=4, sequence length S=2048, number of heads H=32, and we sweep the per-head dimension d from 64 to 256. We observed a staircase pattern (Figure 5). First, multiples of 8 are faster: e.g., d=129 incurs ∼90% higher latency than d=128. Profiling shows that PyTorch uses FlashAttention only when $d \bmod 8 = 0$; otherwise it falls back to the Math kernel. Second, among 8-aligned dimensions, latency rises in a staircase at every 32-dimension boundary. This is caused by FlashAttention-2 (FA2)'s template mechanism: FA2 selects the smallest template $t \geq d$ and assigns a tile shape $B_r \times B_c$ accordingly (Table 2). Crossing a template boundary (e.g., d=128 → 129) halves B_c and increases latency. In Figure 5, alternating shades mark template regions; labels such as "t=96, $B_r \times B_c$=128×64" show the template and tile shape.

3.2 Library Layer

Linear algebra is dispatched to libraries (e.g., cuBLAS), where the same GEMM can be served by different kernels depending on dimensions. We examine this via GEMM C=A·B with

[1] `torch.nn.functional.scaled_dot_product_attention(Q, K, V)`

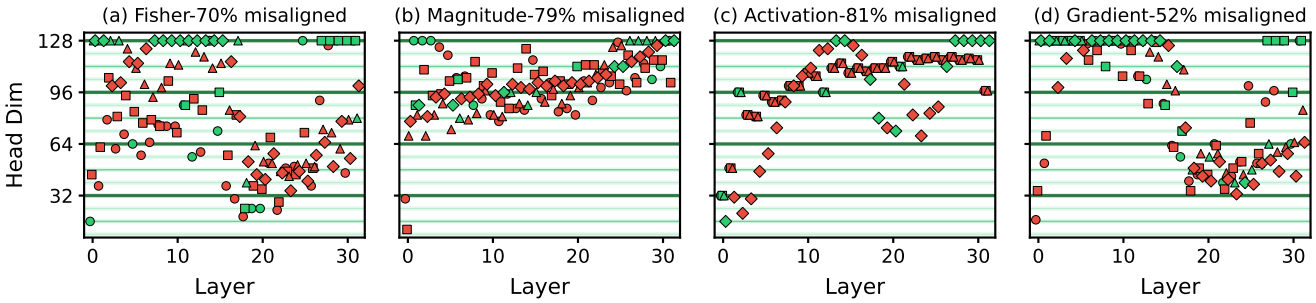


Figure 2. Llama-3-8B at $\rho = 20\%$. Shape: ○=Q Head, □=K Head, △=V Head; color: green=8-aligned, red=misaligned.

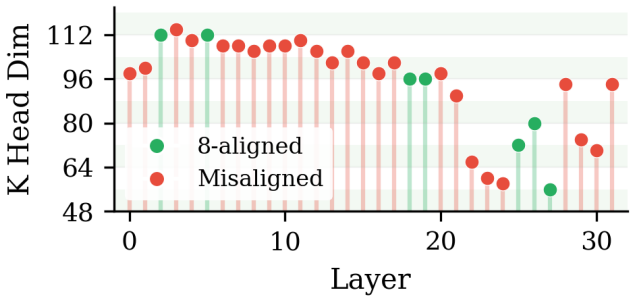


Figure 3. Llama-3-8B after PaLU compression at $\rho = 20\%$.

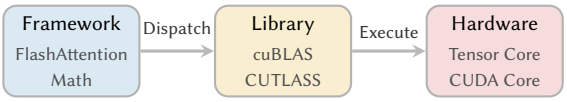


Figure 4. PyTorch SDPA execution stack.

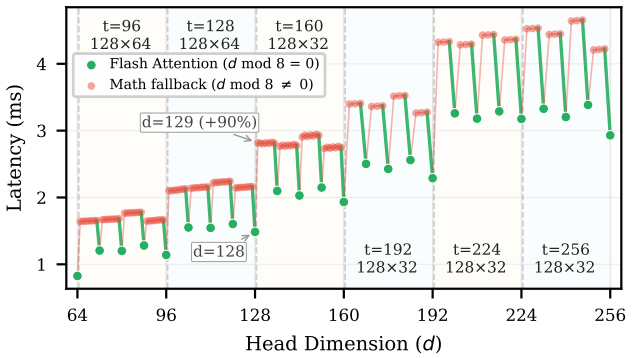


Figure 5. PyTorch SDPA latency across dimensions.

Table 2. FA2 template tiers and performance (B=4, S=2048, H=32).

Region	Template	$B_r \times B_c$	Latency	vs. t=64
d=64	64	128×128	0.74 ms	1.0×
$d \in (64, 96]$	96	128×64	1.12 ms	1.5×
$d \in (96, 128]$	128	128×64	1.47 ms	2.0×
$d \in (128, 160]$	160	128×32	2.00 ms	2.7×
$d \in (160, 256]$	192–256	128×32	2.3–2.9 ms	3–4×

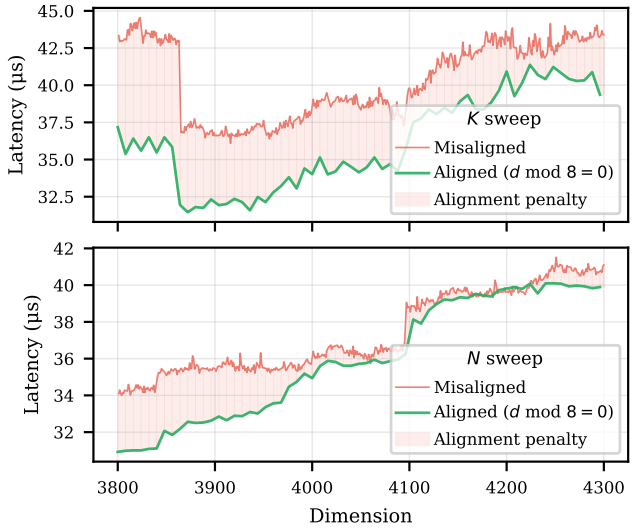


Figure 6. GEMV latency (M=1, stride-1 sweep near 4096).

$A \in \mathbb{R}^{M \times K}$ and $B \in \mathbb{R}^{K \times N}$. We measured GEMM latency with two of (M, N, K) fixed at typical LLM sizes (M=N=2048, K=128) and the third dimension swept from 50% to 100%. Figure 7 shows the results. First, K and N exhibit a clear alignment effect: when the swept dimension satisfies $d \bmod 8 = 0$, latency is lower (e.g., K aligned ~20 μs vs. misaligned 22–26 μs, up to 30% penalty). Second, M and N show *kernel-switching cliffs*: at certain boundaries (e.g., M=1728 → 1729), latency jumps (e.g., ~30%), where the cuBLAS heuristic selected an inefficient kernel. We profiled with Nsight Compute to explain this: when $d \bmod 8 = 0$, cuBLAS invokes its native optimized kernel; otherwise it uses a CUTLASS-generated kernel, which is further divided into align2 (fetching 2 elements at a time) or align1. Table 3 summarizes the three tiers. GEMV (M=1) exhibits a similar but smaller penalty (~12% on K, ~4% on N; Figure 6), consistent with GEMV being memory-bound rather than compute-bound.

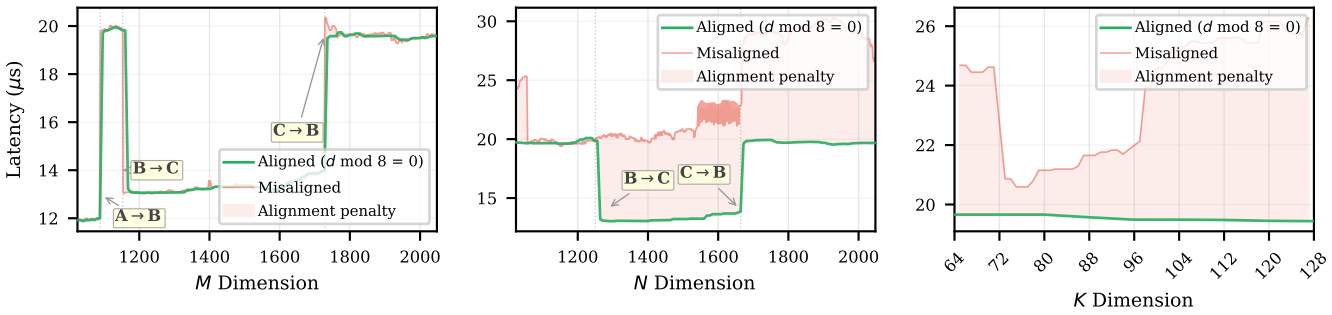


Figure 7. GEMM latency with dimension sweep.

Table 3. cuBLAS GEMM kernel tiers (Nsight Compute).

Tier	Condition	Kernel	MMA Instr.
1	d mod 8 = 0	cuBLAS-native sm80	`mma.m16n8k16`
2	d mod 2 = 0	CUTLASS sm80 align2	`mma.m16n8k16`
3	odd	CUTLASS sm75 align1	`mma.m16n8k8`

Table 4. Full-stack alignment constraints summary.

Level	Mechanism	Constraint	Penalty
Framework	SDPA backend	d%8=0	~90%
Framework	FA2 template	d%32=0	~30%
Library	cuBLAS GEMM	K/N%8=0	~30%
Library	cuBLAS GEMV	K%8=0	~12%
Hardware	TC MMA	K%16=0, N%8=0	~70%
Hardware	L2 sectors	K%16=0	~50%

3.3 Hardware Layer

Beyond framework and library dispatch, misaligned dimensions also cause inefficiency from the hardware level. We use Nsight Compute profiling to isolate two mechanisms. **(1) Tensor Core**: The A100 MMA instruction `mma.m16n8k16` processes tiles of 16×8×16 fp16 elements; dimensions not divisible by these tile sizes leave partial tiles underutilized. A throughput sweep near K, N=4096 confirms: aligned dimensions reach 160–175 TFLOPS while misaligned ones drop to 50–110 TFLOPS, with period-16 in K and period-8 in N matching the tile shape (Figure 8a,b). **(2) Memory**: The A100 L2 Cache operates in 32-byte sectors; for FP16 this requires K mod 16 = 0 for full utilization. Misaligned accesses show ~2× bandwidth loss in microbenchmarks (Figure 8c).

Summary. Table 4 summarizes all constraints. The minimum requirement across all layers is d mod 8 = 0; stricter alignment (mod 16, mod 32) yields further gains. These penalties compound: a single misaligned dimension can trigger a backend fallback, a suboptimal kernel, and underutilized tiles and memory accesses simultaneously.

Algorithm 1 GAC algorithm.

Require: Model $\mathcal{M}$, compressor $\mathcal{F}$, budget B, unit u
Ensure: Aligned dimensions $\{d_i\}_{i=1}^{n}$ with $d_i \in C_i$
Step 1: Unconstrained Compression
$\{d_i^*\}, \{s_i\} \leftarrow \mathcal{F}(\mathcal{M}, B)$ ▷ misaligned dims & scores
Step 2: Dimension Sweep
for each weight $i = 1, \ldots, n$ **do**
  Sweep aligned dims near d_i^* using heuristic constraints from §3
  $C_i \leftarrow$ candidate aligned dims avoiding performance cliffs
end for
Step 3: Constrained Optimization (Knapsack DP)
$B' \leftarrow B/u$ ▷ quantize budget
$D[0..n][0..B'] \leftarrow -\infty$; $D[0][0] \leftarrow 0$
for $i = 1$ to n **do**
  for each $d_{ij} \in C_i$ **do**
    $v_{ij} \leftarrow s_i \cdot (|W_i(d_{ij})| - |W_i^*|)$; $w'_{ij} \leftarrow |W_i(d_{ij})|/u$
    for $b = w'_{ij}$ to B' **do**
      $D[i][b] \leftarrow \max(D[i][b],\ D[i-1][b-w'_{ij}] + v_{ij})$
    end for
  end for
end for
Backtrack from $\arg\max_b D[n][b]$ to get $\{d_i\}$
return $\{d_i\}$

4 GAC: GPU-Aligned Compression

To bridge the gap between compression and alignment, we propose **GAC** (GPU-Aligned Compression), a paradigm that makes budget allocation system-aware, so that fewer parameters translate into real speedup. GAC wraps any dimension-reducing compressor with a post-processing step, re-selecting dimensions to satisfy alignment constraints. Given a model $\mathcal{M}$ with n compressible weights $\mathcal{W}$, a compressor $\mathcal{F}$, and a parameter budget B=$(1-\rho)\,|\mathcal{W}|$ (ρ: compression ratio), GAC produces a fully aligned model in three steps (Figure 9). Algorithm 1 details the GAC steps.

4.1 Step 1: Misaligned Compression

We first apply $\mathcal{F}$ to $\mathcal{M}$ without alignment constraints. $\mathcal{F}$ can be any established dimension-reducing compressor—e.g., ASVD [20] (SVD factorization) or LLM-Pruner [8] (structured pruning). Internally, $\mathcal{F}$ computes a per-weight importance

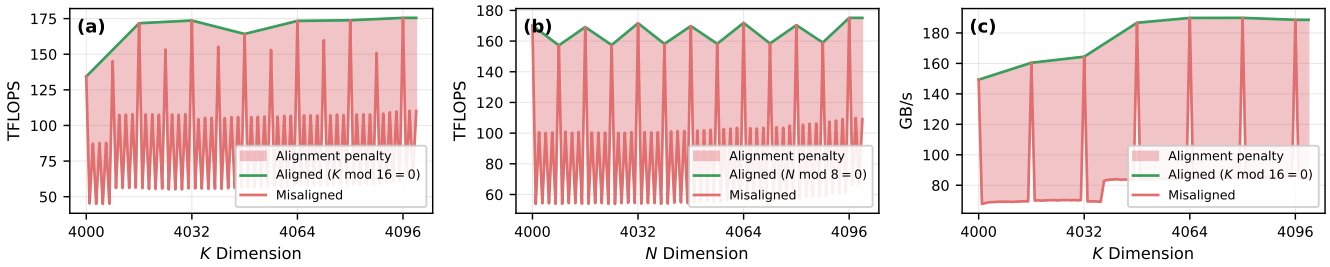


Figure 8. Hardware-level alignment penalties (sweep near 4096): (a,b) Tensor Core throughput, (c) L2 Cache bandwidth.

Figure 9. GAC Pipeline.

score s_i using one of the proxies in Table 1 (e.g., activation magnitude for ASVD, gradient-based Taylor expansion for LLM-Pruner), then allocates dimensions $\{d_i^*\}$ proportionally: higher s_i retains a larger d_i^*. Because s_i and the resulting $\{d_i^*\}$ are continuous, the compressed model is *misaligned*.

4.2 Step 2: Dimension Sweep

A naïve fix would round every d_i^* to the nearest sweet point (e.g. multiple of 8) from the Constraint Table 4. However, we cannot hard-code alignment rounding rules because different operators exhibit different behavior across platforms (e.g., GPU architecture, PyTorch version). A fixed heuristic that works on one platform may miss cliffs or exclude valid dimensions on another.

Instead, GAC selects candidates *empirically*. We use the heuristic constraints (e.g., $d \bmod 8 = 0, d \bmod 16 = 0$) to narrow the search space, then profile the kernel latency at each candidate near d_i^* to verify it avoids performance cliffs on the *actual* platform. This produces a candidate set C_i. For example, given d_i^*=107.3, the sweep yields C_i={96, 104, 112, 128}: dimension 107 is excluded because it triggers a cuBLAS Tier-3 kernel (§3.2), while 104 and 112 both land in Tier-1. Because the sweep is hardware-specific, GAC adapts to different GPU architectures without manual tuning.

4.3 Step 3: Constrained Optimization

With misaligned dimensions $\{d_i^*\}$, importance scores $\{s_i\}$, and candidate sets $\{C_i\}$ in hand, we now select one aligned dimension per weight under the parameter budget. Naïve rounding (e.g., round each d_i^* to the nearest candidate) ignores two factors: (1) different weights have different sensitivities, and (2) rounding up at one weight consumes budget that could be spent elsewhere. We therefore formulate a *multi-choice knapsack* problem:

$$\max_{\{d_i\}} \sum_{i=1}^{n} s_i \cdot (|W_i| - |W_i^*|) \quad \text{s.t.} \quad \sum_i |W_i| \le B, \quad d_i \in C_i \tag{4}$$

where $|W_i|$ is the parameter count of weight W_i at dimension d_i, $|W_i^*|$ at the misaligned dimension d_i^*, and C_i is the candidate set from Step 2. The objective is *asymmetric*: rounding up ($d_i > d_i^*$) preserves information (positive value), rounding down loses it (negative value), each scaled by the per-parameter importance s_i. This lets high-importance weights receive more parameters while low-importance weights absorb the cost.

We solve Eq. 4 via dynamic programming. For each candidate $d_{ij} \in C_i$, define value $v_{ij} = s_i \cdot (|W_i(d_{ij})| - |W_i^*|)$ and cost $w_{ij} = |W_i(d_{ij})|$. The recurrence is:

$$D[i][b] = \max_j \{D[i-1][b - w_{ij}] + v_{ij}\}$$

with complexity $O(n \cdot |C_{\max}| \cdot B')$, where n is the number of weight matrices, $|C_{\max}| = \max_i |C_i|$ the largest candidate set, and B' the quantized budget.

Budget quantization. Naïvely, the DP table has B entries equal to the raw parameter budget, which can reach billions (e.g., $0.85 \times 100 \times 1024^2 \approx 10^8$ for 100 matrices of size 1024×1024 at ρ=15%). However, dimension reduction has a *minimum cost unit*: pruning one column of a $[M, N]$ matrix by 1 adds or removes M parameters. If we further constrain dimensions to multiples of 8, the minimum unit becomes $u = 8 \cdot M_{\min}$, where $M_{\min}$ is the smallest row count across all weights. We quantize costs and budget by u: $w'_{ij} = w_{ij}/u$, $B' = B/u$. This reduces the DP table size dramatically—in the example above, u=8×1024=8192 shrinks the table from $\sim 10^8$ to ~12,500 entries, a reduction of 8000×. In practice, the DP runs in under one second on CPU—negligible compared to the compression itself.

Table 5. Preliminary results on Llama-3-8B (ρ=15%). Measured with batch=1, sequence length =1024.

Method	Align	PPL	PiQA	HSwag	Latency (ms)
Baseline	100%	6.14	0.80	0.50	99.6
ASVD	5%	34.7	0.58	0.28	100.5 (+1%)
ASVD (GAC)	100%	31.3	0.57	0.26	67.1 (−33%)
LLM-Pruner	83%	9.88	0.80	0.49	137.7 (+38%)
LLM-Pruner (GAC)	100%	9.87	0.78	0.47	88.0 (−12%)

5 Evaluation

5.1 Setup

We evaluate on Llama-3-8B with $\rho = 15\%$, using PyTorch 2.9.1, CUDA 12.8, and FP16. Experiments were conducted on two architectures: NVIDIA Ampere (A100-80GB) and Hopper (H100-80GB). Unless stated otherwise, the results are collected from A100. We also repeated a subset of the experiments (due to costs) on H100, which confirmed similar patterns (see Appendix B). We select two representative compressors that alter tensor dimensions in orthogonal ways: **(1) ASVD** [20]: activation-aware SVD ($W \rightarrow A \cdot B$) across all projection weights[2]. **(2) LLM-Pruner** [8]: structured pruning of MLP weights[3]. We compare the uncompressed *baseline* with *Unaligned* (original compression) and *GAC*.

5.2 Implementation

We implement GAC as a proof-of-concept by adding the DP solver on top of the existing ASVD and LLM-Pruner codebases (not yet a standalone framework; see §7). The dimension sweep profiles compression-sensitive kernels (e.g., GEMM, SDPA and GEMV) on the target GPU to build candidate sets (§4.2); the DP solver (§4.3) then selects aligned dimensions. No model architecture changes, no runtime overhead, and no extra inference memory are required—GAC modifies only the dimension allocation before the final compression step.

5.3 Preliminary Results

Table 5 summarizes the end-to-end comparison.

Alignment. ASVD's unconstrained allocation produces 95% misaligned dimensions, while LLM-Pruner produces 17% misaligned dimensions (it only prunes MLP, so attention weights stay aligned). GAC brings both to 100%, snapping every dimension to an aligned candidate via the asymmetric DP objective (Eq. 4).

Accuracy. We report perplexity on WikiText-2 and accuracy on two tasks (PiQA, HSwag; 200 samples each). For ASVD, GAC lowers PPL from 34.7 to 31.3 and the accuracy scores stay comparable (PiQA 0.58→0.57, HSwag 0.28→0.26). For LLM-Pruner, PPL is nearly identical (9.88 vs. 9.87) and downstream accuracy is well preserved (PiQA 0.80→0.78, HSwag 0.49→0.47), confirming that aligned re-selection does not sacrifice quality.

[2]All 32 layers × 7 projections (Q, K, V, O, gate, up, down) = 224 weights.
[3]Layers 3–31 (29/32 layers); gate_proj as pruning root, propagating to up_proj and down_proj.

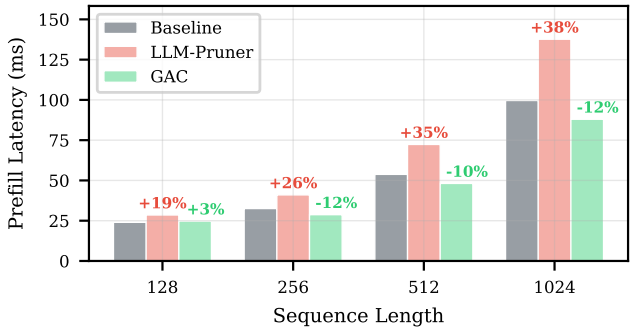


Figure 10. Llama-3-8B latency across sequence lengths.

Latency. We measure prefill latency (batch=1, S=1024; decode latency is left for future work). Despite reducing parameters by 15%, unaligned ASVD shows *no speedup* (100.5 ms vs. 99.6 ms baseline)—the benefit is consumed by misalignment overhead. GAC restores evident speedup (67.1 ms, −33%). For LLM-Pruner, even 83% alignment still incurs +38% latency; GAC eliminates the penalty, achieving 12% speedup over the *uncompressed* baseline. The penalty grows with sequence length (Figure 10): from +19% at S=128 to +38% at S=1024, as longer sequences push GEMMs deeper into the compute-bound regime where alignment (§3.3) dominates.

6 Related Work

System-aware compression. Dimension-reducing compressors such as SVD factorization [2, 18, 20], structured pruning [4, 8, 16], and KV eviction [1, 17, 21] optimize accuracy under a size budget but ignore how the resulting dimensions interact with GPU execution. HALP [15] and HALOC [19] incorporate hardware awareness into CNN compression, but treat latency as a *black-box* signal: they optimize aggregate runtime without isolating *why* certain dimensions are slow, and are tied to specific CNN architectures with no parameter-budget guarantee. GAC instead identifies root causes at three levels (framework dispatch, kernel selection, hardware tile alignment; §3) and constrains dimension selection directly, complementing any upstream compressor while guaranteeing both alignment and parameter budget.

Serving-side mitigations. Serving systems handle misalignment *reactively*. FlashAttention-2 pads to the next template (~30% overhead; §3.1); vLLM [6] rejects unsupported head dimensions. These add overhead or break compatibility. GAC prevents misalignment at compression time, eliminating such workarounds.

7 Limitations and Future Work

Framework automation. Our current implementation adds the GAC DP solver on top of the ASVD and LLM-Pruner codebases. A fully general GAC framework—where users supply only a model name, compression ratio, and compressor, and receive a 100%-aligned model—remains future work.

Model coverage. We evaluate on a single dense model (Llama-3-8B). Extending to more varieties (larger scales, MoE architectures, etc.) would test GAC's generality.

Hardware diversity. Newer GPUs such as Blackwell impose stricter alignment [12–14]; Specialized devices, e.g., DGX Spark and Jetson Nano, add further constraints.

Latency coverage. We report prefill latency at batch=1, S=1024. Covering a wider range of batch sizes and sequence lengths, as well as autoregressive decode latency, would give a fuller performance picture.

Serving-engine compatibility. Our benchmarks use vanilla HuggingFace PyTorch models without any inference-time optimization. Validating GAC under optimized serving engines (e.g., vLLM [6], TensorRT [11]) and DL compilers (e.g., TVM) is needed to confirm alignment gains carry over to production stacks.

References

[1] Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. 2025. PyramidKV: Dynamic KV Cache Compression based on Pyramidal Information Funneling. arXiv:2406.02069 [cs.CL] https://arxiv.org/abs/2406.02069

[2] Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S. Abdelfattah, and Kai-Chiang Wu. 2025. Palu: KV-Cache Compression with Low-Rank Projection. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=LWMS4pk2vK

[3] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 16344–16359. https://proceedings.neurips.cc/paper_files/paper/2022/file/67d57c32e20fd0a7a302cb81d36e40d5-Paper-Conference.pdf

[4] Elias Frantar and Dan Alistarh. 2023. SparseGPT: massive language models can be accurately pruned in one-shot. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) *(ICML'23)*. JMLR.org, Article 414, 15 pages.

[5] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. arXiv:2210.17323 [cs.LG] https://arxiv.org/abs/2210.17323

[6] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 611–626. doi:10.1145/3600006.3613165

[7] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. In *Proceedings of Machine Learning and Systems*, P. Gibbons, G. Pekhimenko, and C. De Sa (Eds.), Vol. 6. 87–100. https://proceedings.mlsys.org/paper_files/paper/2024/file/42a452cbafa9dd64e9ba4aa95cc1ef21-Paper-Conference.pdf

[8] Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. LLM-Pruner: On the Structural Pruning of Large Language Models. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 21702–21720. https://proceedings.neurips.cc/paper_files/paper/2023/file/44956951349095f74492a5471128a7e0-Paper-Conference.pdf

[9] Meta AI. 2024. Llama 3 Model Card. https://github.com/meta-llama/llama3.

[10] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019. Importance Estimation for Neural Network Pruning. arXiv:1906.10771 [cs.LG] https://arxiv.org/abs/1906.10771

[11] NVIDIA. 2024. NVIDIA TensorRT: Programmable Inference Accelerator. https://developer.nvidia.com/tensorrt.

[12] NVIDIA Corporation. 2022. NVIDIA H100 Tensor Core GPU Architecture. White Paper. https://resources.nvidia.com/en-us-hopper-architecture/nvidia-h100-tensor-c.

[13] SemiAnalysis. 2024. NVIDIA Tensor Core Evolution: From Volta To Blackwell. https://newsletter.semianalysis.com/p/nvidia-tensor-core-evolution-from-volta-to-blackwell.

[14] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. 2024. FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-precision. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 68658–68685. doi:10.52202/079017-2193

[15] Maying Shen, Hongxu Yin, Pavlo Molchanov, Lei Mao, Jianna Liu, and Jose M. Alvarez. 2021. HALP: Hardware-Aware Latency Pruning. arXiv:2110.10811 [cs.CV] https://arxiv.org/abs/2110.10811

[16] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2024. A Simple and Effective Pruning Approach for Large Language Models. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=PxoFut3dWW

[17] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. QUEST: query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the 41st International Conference on Machine Learning* (Vienna, Austria) *(ICML'24)*. JMLR.org, Article 1955, 11 pages.

[18] Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. 2025. SVD-LLM: Truncation-aware Singular Value Decomposition for Large Language Model Compression. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=LNYIUouhdt

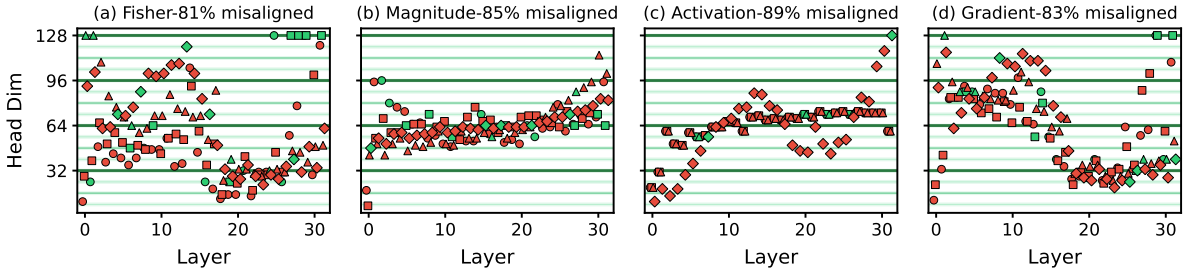
[19] Jinqi Xiao, Chengming Zhang, Yu Gong, Miao Yin, Yang Sui, Lizhi Xiang, Dingwen Tao, and Bo Yuan. 2023. HALOC: hardware-aware automatic low-rank compression for compact neural networks. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence (AAAI'23/IAAI'23/EAAI'23)*. AAAI Press, Article 1175, 9 pages. doi:10.1609/aaai.v37i9.26244

[20] Zhihang Yuan, Yuzhang Shang, Yue Song, Dawei Yang, Qiang Wu, Yan Yan, and Guangyu Sun. 2025. ASVD: Activation-aware Singular Value Decomposition for Compressing Large Language Models. arXiv:2312.05821 [cs.CL] https://arxiv.org/abs/2312.05821

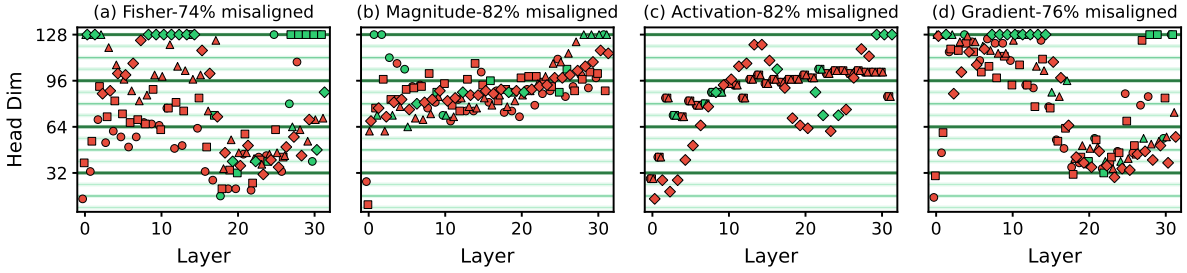
[21] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang "Atlas" Wang, and Beidi Chen. 2023. H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 34661–34710. https://proceedings.neurips.cc/paper_files/paper/2023/file/6ceefa7b15572587b78ecfcebb2827f8-Paper-Conference.pdf

A Dimensional Misalignment Persists Across Compression Ratios

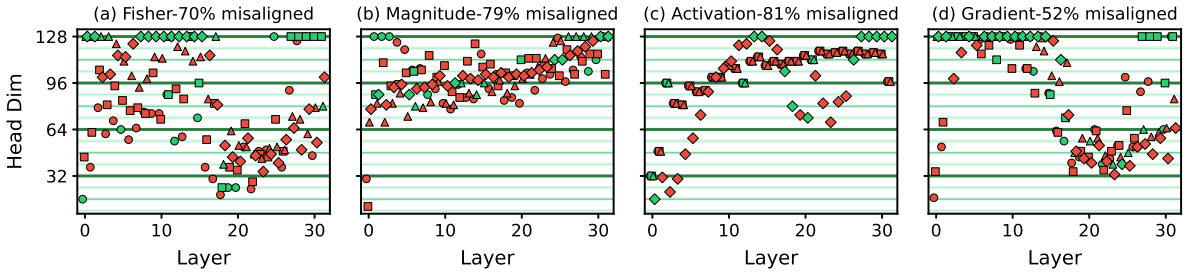
The misalignment problem persists across different compression ratios. Figure 11 shows dimension scatter plots for Llama-3-8B under unconstrained SVD allocation at four compression levels (ρ=10%, 30%, 40%, 50%) using Fisher importance scores. At every ratio, a substantial fraction of dimensions are misaligned, confirming that dimensional misalignment is inherent to importance-based rank allocation, not an artifact of aggressive compression.



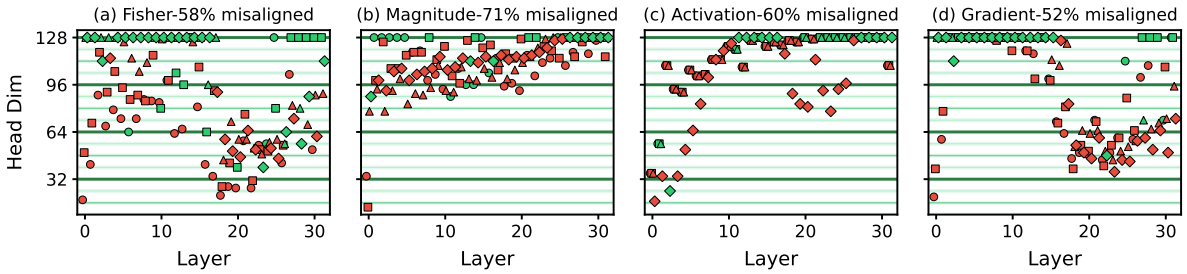
(a) Llama-3-8B, ρ = 50%



(b) Llama-3-8B, ρ = 40%



(c) Llama-3-8B, ρ = 30%



(d) Llama-3-8B, ρ = 10%

Figure 11. Compressed dimension distributions across compression ratios.

B Dimensional Misalignment on H100 GPU

To demonstrate that dimensional misalignment is a fundamental hardware-software co-design issue and not specific to the Ampere architecture, we replicate our key profiling and evaluation experiments on an NVIDIA H100-80GB GPU (Hopper architecture). The Hopper architecture introduces new Tensor Core instructions (e.g., Tensor Memory Accelerator or TMA) and different cache hierarchies, which impose even stricter alignment constraints to achieve peak performance.

Framework-Level Penalties. Figure 12 shows the SDPA latency on H100. Similar to the A100 results, we observe a distinct staircase pattern. However, the penalties for misalignment are even more pronounced. Because the H100 has a significantly higher peak compute throughput, falling back to unoptimized kernels (e.g., the Math backend when $d \bmod 8 \neq 0$) results in a much larger relative slowdown compared to the A100.

Hardware-Level Penalties. Figure 13 details the hardware-level alignment penalties on H100. The impact on Tensor Core throughput (Figure 13a,b) and L2 Cache bandwidth (Figure 13c) follows the same periodic degradation patterns observed on Ampere, confirming that the underlying tile and sector alignment requirements remain critical bottlenecks for irregular dimensions.

End-to-End Latency. Finally, Figure 14 shows the end-to-end prefill latency scaling on H100 for Llama-3-8B. The unaligned compressed models suffer from severe slowdowns, completely negating the theoretical benefits of parameter reduction. In contrast, GAC successfully reduces latency by enforcing hardware-aligned dimensions, especially for larger batch sizes. This demonstrates that our alignment-aware compression paradigm is highly effective across different GPU generations.

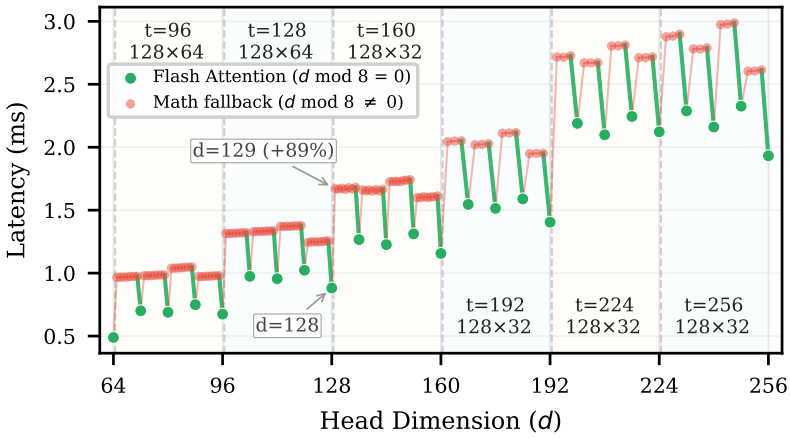


Figure 12. PyTorch SDPA latency across dimensions on H100.

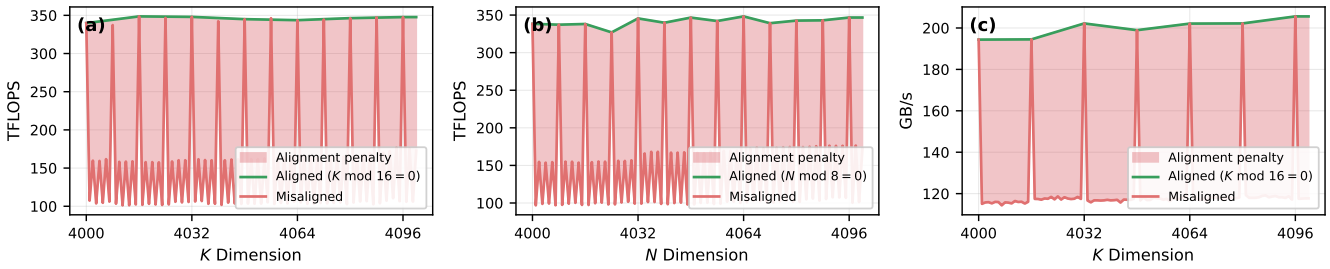


Figure 13. Hardware-level alignment penalties on H100: (a,b) Tensor Core throughput, (c) L2 Cache bandwidth.

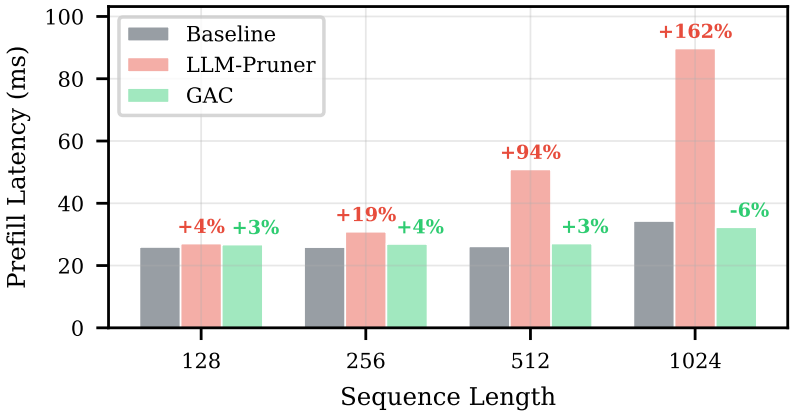


Figure 14. Llama-3-8B latency across sequence lengths on H100.