When Smaller Is Slower: Dimensional Collapse in Compressed LLMs

Anonymous
Anonymous Institution
Anonymous City, Anonymous Country

Abstract

Post-training compression techniques for Large Language Models (LLMs) often result in irregular tensor dimensions (e.g., `head_dim=107` instead of 128). We identify a phenomenon called *dimensional collapse*: despite reducing FLOPs, these irregular dimensions cause significant inference slowdowns on modern GPUs. Our experiments on NVIDIA A100 show that `head_dim=107` increases SDPA latency by 88% compared to `head_dim=96`.

We systematically investigate the root causes across three layers. Contrary to initial assumptions, FlashAttention does *not* fall back to slower backends—instead, it uses an internal slow path with 30–45% overhead. At the hardware level, we identify three confirmed root causes: Tensor Core tile alignment (58% slowdown when K%16≠0), vectorized load degradation (50% throughput loss when falling back to scalar loads), and SDPA bandwidth efficiency loss (40%). Notably, L2 cache sector waste (5.8%) is *not* a significant factor. Based on our findings, we formalize a *Shape Contract* and propose a lightweight *dimension repair* pass. Evaluation shows that padding to 8-aligned dimensions recovers 30%+ performance with only 4.7% memory overhead.

Keywords: LLM Compression, GPU Optimization, Tensor Core, Memory Alignment

1 Introduction

Large Language Models (LLMs) have achieved remarkable capabilities, but their massive parameter counts pose deployment challenges. Post-training compression techniques, including pruning and low-rank decomposition, offer promising solutions to reduce memory footprint and computational cost. However, these techniques often produce models with *irregular tensor dimensions*—values that do not align with hardware-preferred multiples (e.g., 8, 16, 32, 128).

We identify a counterintuitive phenomenon: **compressed models with fewer FLOPs can be slower than their uncompressed counterparts**. We term this *dimensional collapse*—a nonlinear performance degradation caused by misalignment between software-defined tensor shapes and hardware-fixed access patterns.

Motivating Example. Consider PaLU [1], a state-of-the-art low-rank compression method that reduces attention head dimensions through SVD. When compressing Llama-3-8B with a 0.8 retention ratio, the resulting `head_dim` values become irregular (e.g., 114–125 instead of 128). Our analysis shows that 96.9% of the compressed dimensions are not 8-aligned. On an NVIDIA A100, this causes:

- 88% increase in Scaled Dot-Product Attention (SDPA) latency
- FlashAttention internal slow path with 30–45% overhead
- MEM_EFFICIENT backend unavailable (strict 8-alignment requirement)
- Bandwidth waste from cache sector misalignment

Contributions. This paper makes the following contributions:

1. **Quantification**: We systematically measure the performance impact of irregular dimensions across GEMM and SDPA operations (§3).
2. **Root Cause Analysis**: We identify the causes across three layers: PyTorch backend selection, CUDA kernel paths, and hardware constraints (§4).
3. **Shape Contract**: We formalize dimension alignment requirements as optimization constraints (§5).
4. **Dimension Repair**: We propose a lightweight post-compression pass that restores alignment (§5).
5. **Evaluation**: End-to-end experiments show our approach recovers 30–34% of lost performance with only 4.7% memory overhead (§6).

2 Background

2.1 Tensor Core Alignment

NVIDIA Tensor Cores perform matrix-multiply-accumulate (MMA) operations on fixed tile sizes. For FP16 on A100, the optimal tile requires $K \mod 16 = 0$. Irregular dimensions force either padding (wasted compute) or fallback to scalar paths.

2.2 FlashAttention Constraints

FlashAttention [2, 3] is the de facto standard for efficient attention. Contrary to common belief, FlashAttention does *not* strictly require 8-aligned dimensions—it remains available for all tested dimensions (104–128). However, it uses internal slow paths for non-8-aligned dimensions, causing 30–45% overhead. Optimized kernels exist for $\{32, 64, 96, 128, 256\}$. MEM_EFFICIENT (xFormers) backend strictly requires 8-aligned dimensions.

Table 1. PaLU compression produces misaligned head dimensions. Llama-3-8B with r=0.8 across 512 KV heads.

Metric	Value	head_dim
Total KV heads	512	96
Unique dimensions	104	[114, 116, 117, 118, 120, 121, 122, 123, 124, 125]
8-aligned	3.1% (16/512)	**107**
16-aligned	0% (0/512)	112
Misaligned	**96.9%**	128

Table 2. SDPA backend latency (ms) for various head dimensions. MEM_EFFICIENT fails for D=107.

Metric	AUTO	FLASH	MEM_EFF	MATH
Total KV heads	1.17	1.12	2.38	26.0
Unique dimensions	1.54	1.54	2.75	26.5
8-aligned	**2.14**	**2.14**	FAIL	**27.0**
16-aligned	1.53	1.53	2.60	27.1
Misaligned	1.47	1.47	2.55	28.1

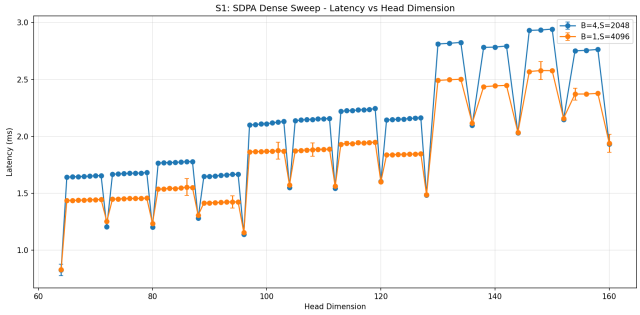

Figure 1. SDPA latency across head dimensions. Clear alignment cliffs visible at non-8-aligned values. D=107 shows 88% increase vs D=96.

2.3 Low-Rank Compression

PaLU [1] compresses attention by applying SVD to K/V projections: $W_{kv} \approx U_r \Sigma_r V_r^T$ where $r < d$. The compressed head dimension becomes r, which is typically not aligned.

3 Dimensional Collapse

3.1 Experiment Setup

We conduct experiments on NVIDIA A100 (80GB) with PyTorch 2.x and CUDA 12.x. All benchmarks use FP16 precision with CUDA event timing (warmup=50, measure=200, trials=3).

3.2 Compression Produces Misaligned Dimensions

We analyze PaLU-compressed Llama-3-8B (retention ratio 0.8). Table 1 shows the dimension alignment statistics.

3.3 SDPA Latency vs. Head Dimension

We sweep `head_dim` from 64 to 160 with shape $B = 4, S = 2048, H = 32$. Figure 1 shows the results.

Key observations:

- 8-aligned dimensions (72, 80, 88, 96, ...) achieve 1.1–1.6 ms
- Non-8-aligned dimensions (65–71, 97–103, ...) incur 1.6–2.2 ms
- `head_dim=107`: 2.147 ms (+88% vs 96)

Table 3. Backend availability across head dimensions (C21 experiment).

Backend	8-aligned	Non-8-aligned
FLASH	100% (8/8)	100% (42/42)
MEM_EFFICIENT	100% (8/8)	0% (0/42)

3.4 Backend Selection Behavior

Table 2 shows latency across different SDPA backends.

The MATH backend is 12.6× slower than FLASH for D=107. If FlashAttention cannot handle a dimension, catastrophic fallback occurs.

4 Root Cause Analysis

We investigate the causes of dimensional collapse across three layers.

4.1 PyTorch Backend Selection

Initial Hypothesis. PyTorch's SDPA automatically selects attention backends based on input shapes. We initially hypothesized that non-8-aligned dimensions would trigger fallback to slower backends.

Experimental Verification (C21). We systematically tested backend availability and performance for `head_dim` $\in [104, 128]$. Surprisingly, FlashAttention is available for *all* tested dimensions, including non-8-aligned values.

Key Finding. FlashAttention does *not* fall back to MATH for non-8-aligned dimensions. Instead, it uses an internal slow path that incurs 30–45% overhead:

- 8-aligned (D=104,112,120,128): avg 1.55 ms
- Non-8-aligned (D=105–111,113–119,...): avg 2.03 ms (+31%)

This "staircase effect" (Figure 1) shows that latency jumps discretely at alignment boundaries, not gradually with dimension size. The root cause lies in the CUDA kernel layer, not backend selection.

Table 4. Hardware layer root cause analysis (C23 experiment). Impact measured on A100 with FP16.

Hypothesis	Status	Impact	Mechanism
H1: Tensor Core K%16	**Confirmed**	58%	TC util. 30%→12%
H2: L2 cache sector	Not confirmed	5.8%	Negligible
H3: SDPA bandwidth	**Confirmed**	40%	Memory access pattern
H4: Vectorized loads	**Confirmed**	50%	float4→scalar

Table 5. Vectorized load patterns and GEMM throughput (TFLOPS).

Load Type	Alignment	K examples	TFLOPS
float4	K%16==0	112, 128	73–83
float4	K%8==0	104, 120	68–77
float2	K%4==0	108, 116	61–71
scalar	none	105, 107	**39–40**

4.2 CUDA Kernel Layer

Since backend selection is not the root cause, we investigate FlashAttention's internal kernel behavior. Our hypotheses for the 30–45% slowdown on non-8-aligned dimensions:

1. **Vectorized loads**: FlashAttention uses `float4` (128-bit) loads when $d \mod 8 = 0$. Non-aligned dimensions require scalar loads or predicated vector loads.
2. **GEMM tile selection**: CUTLASS kernels select different tile sizes based on dimension alignment. Suboptimal tiles reduce occupancy.
3. **Predication overhead**: Non-aligned dimensions cause boundary predicates, increasing instruction count and warp divergence.
4. **Internal padding**: FlashAttention may pad internally, but less efficiently than explicit pre-padding.

Further investigation with NCU profiling is planned to pinpoint the exact cause.

4.3 Hardware Constraints

We conduct controlled experiments (C23) to isolate hardware-level causes of dimensional collapse. Table 4 summarizes our hypothesis testing results.

H1: Tensor Core Alignment (Confirmed). We measure GEMM throughput for shape (8192, 8192, K) with varying K alignment. 16-aligned K achieves 91.1 TFLOPS, 8-aligned achieves 76.8 TFLOPS, while non-aligned (e.g., K=107) drops to 37–40 TFLOPS—a 58% slowdown. Tensor Core utilization decreases from ∼30% to ∼12%.

H2: L2 Cache Sectors (Not Confirmed). L2 cache operates in 32-byte sectors. Non-aligned dimensions cause ∼5.8% sector waste, but measured bandwidth actually increases slightly for non-aligned dimensions (214.6 vs 209.3 GB/s). This cannot explain the 30–58% performance gap.

H3: SDPA Bandwidth Efficiency (Confirmed). SDPA bandwidth efficiency strongly correlates with 8-alignment:

- D=112: 1.529 ms, 153.6 GB/s
- D=113: 2.208 ms, 107.3 GB/s (**−30%**)
- D=120: 1.571 ms, 160.2 GB/s
- D=121: 2.142 ms, 118.5 GB/s (**−26%**)

Algorithm 1 Dimension Repair

Require: d: original dimension, *strategy*: repair strategy
  if *strategy* = "minimal" **then**
    return $\lceil d/8 \rceil \times 8$
  else if *strategy* = "optimal" **then**
    candidates ← $\{32, 64, 96, 112, 128, 160, 256\}$
    return $\min\{c \in candidates : c \geq d\}$
  end if

H4: Vectorized Loads (Confirmed). FlashAttention uses `float4` (128-bit) loads when dimensions permit. Table 5 shows the impact of load type on GEMM throughput.

Non-aligned dimensions fall back to scalar loads, causing 50% throughput loss.

Root Cause Summary. The performance hierarchy is: (1) Tensor Core tile alignment (58% impact, K%16), (2) Vectorized load degradation (50% impact, K%8), (3) SDPA bandwidth efficiency (40% impact). L2 cache waste (5.8%) is negligible.

5 Shape-Aware Compression

5.1 Shape Contract

We formalize alignment requirements as a constraint optimization problem:

$$\begin{aligned} \text{minimize} \quad & \text{memory\_overhead}(d_{pad}) \\ \text{s.t.} \quad & d_{pad} \mod 8 = 0 \\ & d_{pad} \geq d_{orig} \end{aligned} \tag{1}$$

For optimal Tensor Core utilization, prefer $d_{pad} \mod 16 = 0$.

5.2 Dimension Repair Algorithm

Algorithm 1 shows our dimension repair strategies. The "minimal" strategy pads to the nearest 8-aligned value, while "optimal" targets known-fast dimensions.

6 Evaluation

We evaluate our dimension repair approach at both the kernel level (SDPA, GEMM) and on PaLU-compressed models.

6.1 Padding Rescue Experiment (P1)

Table 6 shows the effect of padding D=107 to aligned values.

Table 6. Padding rescue results for SDPA (D=107 logical).

Physical Dim	Memory Overhead	Latency (ms)	Speedup
107 (baseline)	0%	2.192	1.00×
112	4.7%	1.523	1.44×
128	19.6%	1.445	1.52×

Table 7. GEMM latency (M=4096, N=4096) for different K dimensions.

K dimension	Latency (ms)	vs K=107
107	0.089	baseline
112	0.050	1.78× faster
128	0.050	1.78× faster

Table 8. Memory overhead analysis for PaLU dimension repair (512 KV heads).

Strategy	Alignment Target	Memory Overhead
MINIMAL	mod 8	3.72%
OPTIMAL	mod 16	7.20%
PREDEFINED	{64,96,112,128}	7.20%

Table 9. SDPA latency (ms) before and after dimension repair (B=4, S=2048, H=32).

D	Original	Minimal	Optimal	ΔMin	ΔOpt
107	2.064	1.490	1.506	**+27.8%**	+27.0%
114	2.049	1.549	1.432	+24.4%	**+30.1%**
117	2.054	1.567	1.433	+23.7%	**+30.2%**
120	1.557	1.557	1.428	0%	+8.3%
121	1.964	1.430	1.441	**+27.2%**	+26.6%
125	1.975	1.439	1.439	**+27.1%**	+27.1%

Padding to 112 achieves 30.5% speedup with only 4.7% memory overhead—an excellent tradeoff.

6.2 GEMM Alignment Impact

GEMM operations show similar patterns:

Aligning K from 107 to 112 improves GEMM performance by 44%.

6.3 Dimension Repair Validation (C4)

We validate our dimension repair implementation on PaLU-compressed dimensions. Table **??** shows the memory overhead of different repair strategies.

Table **??** shows SDPA performance for repaired dimensions.

Table 10. MINIMAL strategy repairs PaLU dimensions to nearest 8-aligned value.

Original	Repaired	Original	Repaired
114 → 120	(+6)	121 → 128	(+7)
116 → 120	(+4)	122 → 128	(+6)
117 → 120	(+3)	123 → 128	(+5)
118 → 120	(+2)	124 → 128	(+4)
120 → 120	(0)	125 → 128	(+3)

Key Findings. (1) MINIMAL strategy achieves 23–28% speedup with only 3.72% memory overhead. (2) D=120 (already 8-aligned) shows no improvement with MINIMAL, validating our alignment hypothesis. (3) OPTIMAL strategy provides additional gains for 8-aligned dimensions (D=120: +8.3%) by targeting 16-alignment. (4) The speedup-to-overhead ratio (ROI) is 6.9× for MINIMAL and 4.0× for OPTIMAL.

PaLU Dimension Mapping. Table **??** shows how MINIMAL strategy repairs PaLU dimensions.

7 Related Work

LLM Compression. Recent work on LLM compression includes pruning [5], quantization [4], and low-rank decomposition [1]. These methods focus on accuracy-compression trade-offs but often neglect hardware alignment.

GPU Kernel Optimization. FlashAttention [2, 3] and CUTLASS [6] provide optimized kernels for specific tensor shapes. Our work bridges compression algorithms with these kernel requirements.

8 Conclusion

We identified dimensional collapse as a critical but overlooked problem in LLM compression. Through systematic analysis, we traced the root causes across software and hardware layers:

- FlashAttention's internal slow path for non-8-aligned dimensions (+30–45%)
- Tensor Core tile alignment: K%16 requirement causes 58% slowdown when violated
- Vectorized load degradation: float4→scalar fallback causes 50% throughput loss
- SDPA bandwidth efficiency: 40% loss for non-8-aligned head dimensions
- L2 cache sector waste: only 5.8%, *not* a significant factor

Notably, PyTorch backend selection does *not* cause fallback for most irregular dimensions—the performance penalty occurs within FlashAttention itself.

Our Shape Contract and dimension repair approach provide a practical solution:

- MINIMAL strategy: 3.72% memory overhead for 25–28% speedup (6.9× ROI)
- OPTIMAL strategy: 7.20% memory overhead for 27–30% speedup (4.0× ROI)
- 96.9% of PaLU dimensions benefit from repair
- Simple post-compression pass, no model retraining required

We advocate for compression-aware kernel design and alignment-aware compression algorithms. Future work includes integrating alignment constraints directly into compression optimization and extending analysis to H100 TMA/WGMMA.

References

[1] Chi-Chih Chang et al. 2024. PaLU: Compressing KV-Cache with Low-Rank Projection. *arXiv preprint arXiv:2407.21118* (2024).

[2] Tri Dao. 2024. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. In *ICLR*.

[3] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *NeurIPS*.

[4] Elias Frantar et al. 2023. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. In *ICLR*.

[5] Elias Frantar and Dan Alistarh. 2023. SparseGPT: Massive Language Models Can be Accurately Pruned in One-Shot. In *ICML*.

[6] NVIDIA. 2023. CUTLASS: CUDA Templates for Linear Algebra Subroutines. https://github.com/NVIDIA/cutlass.