Demystifying Numerical Instability in LLM Inference: Achieving Reproducible Inference for Mission-Critical Tasks with HEAL

Zhenting Zhu[1], **Lucas Thai**[1], **Shan Yu**[1], **Yicheng Liu**[1], **Yifan Qiao**[2], **Chenxi Wang**[3], **Harry Xu**[1], **Junyi Shu**[1]

[1]UCLA [2]UC Berkeley [3]University of Chinese Academy of Sciences

Abstract

As Large Language Models (LLMs) deploy into mission-critical domains (*e.g.*, finance, medicine, and law), output reproducibility has become a strict system requirement. While practitioners use greedy decoding to eliminate algorithmic stochasticity, empirical deployments with 16-bit precisions still exhibit catastrophic output divergence across heterogeneous GPUs. Through SASS-level profiling, we reveal that this inconsistency is fundamentally driven by truncation errors introduced during downcasting at kernel boundaries. However, achieving reproducibility via a global FP32 pipeline incurs prohibitive system penalties: bypassing 16-bit hardware accelerators hurts compute efficiency, while upcasting the KV cache doubles memory overhead. To bridge this gap, we propose **H**ybrid **E**rror **AL**leviation (**HEAL**), a targeted intervention that approximates FP32 precision while resolving hardware constraints through two targeted mechanisms. First, recognizing that floating-point formats underutilize their bit-width for Q, K, V tensors, HEAL applies INT16 quantization that preserves numerical stability without expanding the KV cache footprint. Second, HEAL synthesizes high-precision matrix multiplications via an algebraic error compensation strategy, executing entirely on high-throughput 16-bit Tensor Cores. To evaluate our approach practically, we introduce MCR-Bench, a benchmark targeting reproducibility in mission-critical tasks. HEAL achieves the same level of reproducibility on downstream tasks as the FP32 baseline while reducing the performance overhead by up to 6.0×.

1 Introduction

As Large Language Models (LLMs) transition from creative assistants to the backbone of mission-critical applications—such as enterprise-grade services [44, 54, 67], finance [21, 36], medical diagnosis [6, 19, 63], and regulatory legal compliance [5, 58, 71]—the primary requirement shifts from mere fluency to strict system reliability. In these high-stakes workflows, a fundamental expectation is reproducibility: a fixed input must yield a functionally identical outcome. To achieve this, practitioners typically employ greedy decoding (temperature=0), a setting universally assumed to remove stochasticity from the generation process.

However, the infrastructure supporting these models makes this guarantee exceedingly difficult to uphold. Modern datacenters are inherently heterogeneous, frequently mixing different generations and vendors of accelerators due to rapid procurement cycles and massive compute demands. For instance, older generations of NVIDIA GPUs remain ubiquitous [33, 62], while leading model providers increasingly integrate non-NVIDIA hardware to scale capacity and mitigate vendor lock-in [2, 3, 47, 48]. Consequently, LLM inference remains non-deterministic even under greedy decoding. When identical inference code executes across varied hardware, the non-associativity of floating-point arithmetic—combined with dynamically shifting software configurations (*e.g.*, varying attention

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

backends or batch sizes)—causes generated tokens to diverge [26, 54, 60]. In domains requiring exact numerical outputs (*e.g.*, specific ICD-10 medical codes [15], or boolean legal determinations), a single divergent token can invalidate the entire semantic pipeline.

To secure the practical, task-level reproducibility required by mission-critical applications, existing mitigation strategies attempt to enforce strict bit-wise determinism—ensuring every generated token matches exactly across runs. These approaches generally fall into two categories, both of which present an unacceptable dilemma for live production environments. One line of work enforces deterministic floating-point reduction orders, through framework-level deterministic modes, hardware-specific features such as CUDA backward compatibility, or specialized deterministic kernels and inference engines [13, 26, 31, 50, 60, 72]. While these methods achieve bit-level consistency, they restrict deployment to narrow execution paths, disable modern hardware features, sacrifice throughput, and fail to generalize across disparate GPU vendors or generations. An alternative approach seeks to suppress numerical instability through higher-precision computation, performing all calculations in FP32 [70]. Although this approach effectively eliminates downcasting noise and is theoretically generalizable, practical barriers make its adoption prohibitively expensive. Serving entirely in FP32 severely degrades generation speed, increasing the Time Per Output Token (TPOT) by up to 13.1×, rendering it economically impractical.

Key Insight. This work challenges the assumption that the all-or-nothing cost of full high-precision pipelines is truly required to achieve robust, deployment-ready reproducibility. To systematically investigate the root causes of non-reproducibility and the structural barriers to efficient mitigation, we establish comprehensive baselines across standard precision formats. Through SASS-level profiling, we find that instability is not driven by internal kernel arithmetic (which predominantly operates in FP32), but by *the truncation of high-precision registers during downcasting at kernel boundaries*. Because extreme numerical overflows are rare during inference, it is these microscopic losses of fractional precision that accumulate over the network depth. Eventually, this end-to-end truncation error shifts the final log probability enough to alter the top-1 token selection, directly causing non-reproducibility.

Crucially, our analysis reveals that we do not need strict bitwise parity to achieve reproducibility. Instead, we only need to bound numerical noise below the threshold that triggers result flipping. However, naïve attempts to enforce this bounded precision through FP32 upcasting run into two fatal hardware walls. First, for `Attention`, scaling precision introduces a memory bottleneck. Upcasting the Key-Value (KV) cache to 32-bit strictly doubles its memory footprint, severely degrading the performance. Second, for `GEMM`, scaling up to higher precision triggers a compute bottleneck. Modern AI accelerators achieve high throughput by relying on specialized hardware units (*e.g.*, Tensor Cores) that structurally require 16-bit inputs. Upcasting inputs to FP32 to preserve precision forces the hardware to fall back to standard execution units, causing a massive performance regression.

HEAL. To resolve this fundamental tradeoff between reproducibility and hardware efficiency, we propose **H**ybrid **E**rror **AL**leviation (**HEAL**), a targeted intervention that achieves FP32-level reproducibility while effectively bypassing both the `Attention` and `GEMM` bottlenecks. To resolve the `Attention` bottleneck, HEAL avoids full FP32 cache upcast. Recognizing that floating-point formats underutilize their bit-width for Q, K, V tensors, HEAL applies a targeted INT16 quantization. This achieves the necessary fractional precision to prevent divergence without expanding the physical KV cache footprint. To address the `GEMM` bottleneck, HEAL synthesizes high-precision matrix multiplications without bypassing 16-bit hardware accelerators. It employs an algebraic error compensation strategy, decomposing the input activations into a primary 16-bit value and a residual error term. By evaluating the matrix multiplication for both components using standard 16-bit inputs, the system captures the truncation error while executing entirely on high-throughput Tensor Cores.

To evaluate our approach practically, we introduce MCR-Bench (Mission-Critical Reproducibility Benchmark), a benchmark targeting reproducibility in domains where outcome consistency is paramount. Our evaluation across a diverse spectrum of modern LLMs demonstrates that HEAL achieves the same level of downstream task reproducibility as the FP32 baseline. With our hybrid error alleviation strategy, HEAL incurs only a 70%–120% TPOT overhead, a stark contrast to the severe 4.2–13.1× slowdown exhibited by the FP32 pipeline [70] on various GPU and model types.

In summary, this paper makes the following contributions:

- We demonstrate that the intrinsic non-reproducibility is fundamentally driven by boundary truncation errors, and establish that robust reproducibility requires bounding this noise below a result flipping threshold (§2).
- We introduce Hybrid Error ALleviation (HEAL), a two-pronged solution that bypasses hardware penalties by safeguarding `Attention` via INT16 quantization and synthesizing high-precision `GEMM` via algebraic error compensation (§3).
- We introduce MCR-Bench, demonstrating that HEAL achieves FP32-level reproducibility for mission-critical downstream tasks across varied hardware types while reducing the performance overhead by up to 6.0× (§4).

The source code of HEAL and the MCR-Bench data are publicly available at `https://github.com/ucla-systems/HEAL`.

2 The Anatomy of Non-reproducibility

To understand why standard 16-bit deployments (*e.g.*, BF16) fail to provide the deterministic reproducibility required by mission-critical applications, we must isolate the structural mechanisms driving end-to-end divergence. In this section, we systematically trace the lifecycle of numerical errors in reverse: starting from the macroscopic failure where final log-probabilities shift enough to change the top-1 token, tracing back to observe how these errors accumulate and propagate across network layers, and finally isolating their microscopic origin at the hardware level.

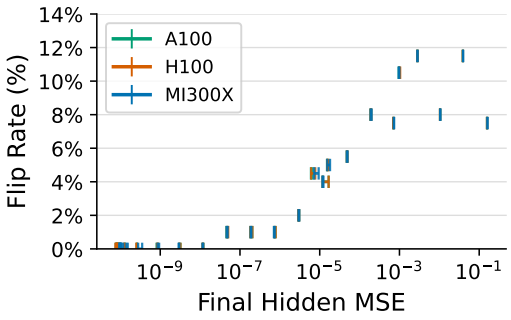


Figure 1: Answer flip rate vs. total numerical error.

To ensure our findings represent fundamental systemic behaviors rather than artifacts of a specific architecture, our diagnostic analysis spans five widely deployed open-source models[1]: Qwen3-8B, 14B and 32B [68], Llama-3.1-8B-Instruct [22], and DeepSeek-R1-Distill-Qwen-14B [24]. Furthermore, to measure numerical error accurately without the confounding effects of autoregressive branching, we enforce strict execution path alignment via Teacher Forcing [65]. Specifically, we utilize an NVIDIA H100 executing a full FP32 pipeline as our deterministic anchor, tracking the layer-by-layer divergence of low-precision executions against this anchor.[2]

2.1 The Macroscopic Failure: Result Flipping

We begin at the end of the inference pipeline. Because strict reproducibility is a baseline requirement for mission-critical applications, these systems naturally default to greedy decoding. In this deterministic setting, the model's final decision relies on a simple argmax operation over the output logits: token $= \arg\max(p(t))$. Therefore, numerical error only manifests as non-reproducibility if it accumulates enough magnitude to bridge the probability margin between the top-1 and top-2 tokens.

Crucially, this structural reliance on argmax creates a highly non-linear relationship between the accumulated numerical noise and the likelihood of result flipping. The network exhibits a distinct *divergence threshold*. To quantify the accumulated noise just before the final decision, we measure the Final Hidden MSE, defined as the mean squared error in the representations immediately before the vocabulary projection matrix (i.e., the normalized final hidden layer output). As shown in Figure 1, which tracks the probability of the choice being flipped on MedQA [34] against this Final Hidden MSE, we observe two distinct operational regimes:

- **The Benign Regime (Sub-Threshold):** When the accumulated error remains small, it is safely absorbed by the inherent probability margin between the top candidates. The downstream mission-

[1]This section presents the results for Qwen3-14B. The results for the remaining models are detailed in Appendix B.
[2]For more details on the experiment setup, please refer to Appendix A and B.

critical logic remains entirely intact, demonstrating the model's natural tolerance to minor numerical fluctuations.

- **The Catastrophic Regime (Super-Threshold):** However, once the accumulated error breaches a critical threshold, the probability of a choice flip surges dramatically. When internal errors are heavily amplified, they inject massive absolute deviations that overcome the probability margin. This results in catastrophic divergence, such as misdiagnosing a life-threatening acute condition as a benign symptom due to shifting to a distractor option.

Consequently, this non-linear dynamic dictates that strict bitwise parity with an FP32 reference is an unnecessarily strict system requirement. To guarantee downstream functional reproducibility, it is sufficient to maintain the accumulated numerical errors safely within the benign, sub-threshold regime.

2.2 The Propagation and Accumulation of Error

If the network can naturally tolerate minor background noise, how does the numerical error accumulate the magnitude required to breach the catastrophic divergence threshold? To answer this, we abstract the standard LLM inference procedure and track the propagation of error layer-by-layer.

Take the forward pass of Qwen3CausalLM (Algorithm 1) as an example. For the decoder layer i, there are mainly three steps: first compute $\mathbf{K}_i, \mathbf{V}_i$ and update the KV cache, then compute $\mathbf{A}_i$, and finally compute $\mathbf{H}_i$ as next layer's input. Suppose we use a lower precision p, and let $\mathcal{F}(\mathbf{H}_i, \mathbf{C}^{(i)}, \mathbf{W}_o^{(i)})$ be the mapping from pre-attention tensor $\mathbf{H}_i$ to post-attention tensor $\mathbf{A}_i$, and $\mathcal{F}_p$ being its low precision counterpart, and let $\mathcal{Q}_p$ denote the function truncating values to precision p. We can then formally decompose the error between the exact $\mathbf{A}_i$ and the computed $\hat{\mathbf{A}}_i$ as:

Algorithm 1: Qwen3CausalLM Forward Pass

Input: Tokens $\mathbf{T}$, KV Caches $\mathbf{C}$, Weights $\mathbf{W}$, Metadata $\mathcal{M}$
Output: Next tokens $\mathbf{T}_{next}$
$\mathbf{H}_0 \leftarrow$ Embedding($\mathbf{T}$);
for $i = 0$ **to** $L - 1$ **do**
  q, k, $\mathbf{V}_i \leftarrow$ GEMM $\left(\text{Norm}(\mathbf{H_i}), \mathbf{W}_{qkv}^{(i)}\right)$;
  $\mathbf{Q}_i, \mathbf{K}_i \leftarrow$ RoPE (QKNorm(q, k)); $\mathbf{C}^{(i)} \leftarrow \mathbf{C}^{(i)} + \{\mathbf{K}_i, \mathbf{V}_i\}$;
  $\mathbf{A}_i \leftarrow \mathbf{H}_i +$ GEMM $\left(\text{Attention}(\mathbf{Q}_i, \mathbf{C}^{(i)}, \mathcal{M}), \mathbf{W}_o^{(i)}\right)$;
  g, u $\leftarrow$ GEMM $\left(\text{Norm}(\mathbf{A}_i), \mathbf{W}_{gate_up}^{(i)}\right)$;
  $\mathbf{H}_{i+1} \leftarrow \mathbf{A}_i +$ GEMM $\left(\text{SiLU(g)} \odot \text{u}, \mathbf{W}_{down}^{(i)}\right)$;
end
$\mathbf{L} \leftarrow$ GEMM(ExtractLastTokens(Norm($\mathbf{H}_L$), $\mathcal{M}$), $\mathbf{W}_{lm_head}$);
return *Sample*($\mathbf{L}$);

$$\hat{\mathbf{A}}_i - \mathbf{A}_i = \mathcal{Q}_p\left(\mathcal{F}_p(\hat{\mathbf{H}}_i, \hat{\mathbf{C}}^{(i)}, \mathbf{W}_o^{(i)})\right) - \mathcal{F}(\mathbf{H}_i, \mathbf{C}^{(i)}, \mathbf{W}_o^{(i)}) \tag{1}$$

$$= \mathcal{Q}_p\left(\mathcal{F}_p(\hat{\mathbf{H}}_i, \hat{\mathbf{C}}^{(i)}, \mathbf{W}_o^{(i)})\right) - \mathcal{F}(\hat{\mathbf{H}}_i, \hat{\mathbf{C}}^{(i)}, \mathbf{W}_o^{(i)}) + \quad (\epsilon_{\text{new}}) \tag{2}$$

$$\mathcal{F}(\hat{\mathbf{H}}_i, \hat{\mathbf{C}}^{(i)}, \mathbf{W}_o^{(i)}) - \mathcal{F}(\hat{\mathbf{H}}_i, \mathbf{C}^{(i)}, \mathbf{W}_o^{(i)}) + \quad (\epsilon_{\text{kv}}) \tag{3}$$

$$\mathcal{F}(\hat{\mathbf{H}}_i, \mathbf{C}^{(i)}, \mathbf{W}_o^{(i)}) - \mathcal{F}(\mathbf{H}_i, \mathbf{C}^{(i)}, \mathbf{W}_o^{(i)}). \quad (\epsilon_{\text{old}}) \tag{4}$$

There are three distinct structural sources: the newly injected error ϵ_{new} contributed by the rounding error of $\mathcal{F}_p$, the horizontal error ϵ_{kv} inherited from past tokens via the KV cache, and the vertical error ϵ_{old} propagated upward from prior layers via the residual stream. This derivation also applies to $\mathbf{K_i}, \mathbf{V_i}, \mathbf{H_{i+1}}$ and the final $\mathbf{L}$, in which there is no ϵ_{kv}. Numerical instability stems from floating-point rounding errors, which is contributed by three mappings within each layer: $\mathcal{F}^{(\text{kv})} : \mathbf{H} \to \mathbf{K}, \mathbf{V}$, $\mathcal{F}^{(\text{attention})} : \mathbf{H} \to \mathbf{A}$, $\mathcal{F}^{(\text{mlp})} : \mathbf{A} \to \mathbf{H}$. Consequently, we isolate the relative contribution of each source, and attribute the final error onto different layers, as shown in Figure 2. Please find our detailed experiments and discussions regarding this process in Appendix B.2.

This layer-wise tracking reveals two critical operational dynamics of LLM inference: i) *Non-Linear Accumulation*: As shown by the red line in Figure 2, the total accumulated Mean Squared Error (MSE) exhibits exponential growth. The numerical noise magnifies multiplicatively as it propagates deeper into the network, pushing the final log-probabilities toward the catastrophic divergence threshold. ii) *Structural Imbalance*: Crucially, the injection of this catastrophic error is not ubiquitous. The bar chart in Figure 2 demonstrates that layers have different contributions within the whole model.

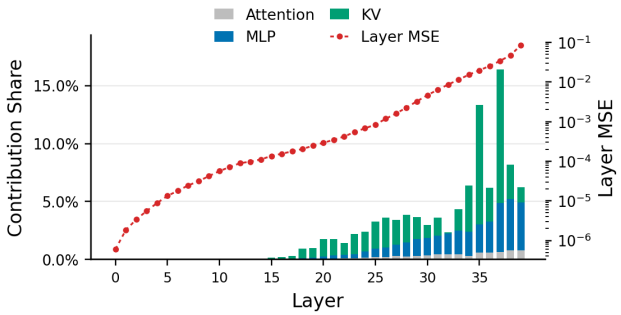


Figure 2: The contribution of error sources across the network.

This structural imbalance dictates a straightforward mitigation strategy. While the error amplification is concentrated in specific bottlenecks, we observe that even 1% of the MSE accumulated under native BF16 strictly exceeds the divergence threshold. Consequently, simply upcasting the one or two highest peaks is mathematically insufficient; to preserve end-to-end reproducibility, we must actively apply our upcasting interventions across the majority of the network's layers. The initial layers, whose error contributions are minor, can optionally remain in lower precision to recover throughput.[3]

2.3 The Microscopic Origin: Boundary Truncation

Having established that catastrophic divergence is driven by the structural amplification of baseline errors within specific layers, we must isolate the microscopic origin of these errors. What exactly generates the new error injected at every layer?

A common misconception is that standard 16-bit inference implies low-precision arithmetic throughout the entire execution pipeline. If true, the mathematical operations themselves would be the primary source of instability. To test this assumption, we profiled the NVIDIA SASS (Streaming ASSembly) instructions of the primary inference kernels on an NVIDIA A100, encompassing modern Attention implementations (FlashAttention [56] and TritonAttention [53]).

Table 1: Breakdown of internal floating-point arithmetic instructions for primary kernels in LLM inference.

Kernel	Instructions
ArgmaxReduce	**HFMA2.MMA** (100.0%)
FlashFwdSplitKV	**HMMA.16816.F32.BF16** (43.2%), FADD.FTZ (11.1%), FFMA.FTZ (11.1%), FMNMX.FTZ (11.1%), FMUL.FTZ (11.5%)
FusedAddRMSNorm	**HFMA2.BF16_V2** (16.3%), FADD (49.3%), FFMA (9.2%), FMUL (24.8%), MUFU.RCP (0.2%)
GEMM	**HMMA.16816.F32.BF16** (93.0% - 97.6%), FADD (1.3% - 2.3%), FFMA (2.3%), FMUL (2.4% - 5.9%)
GPUIndexKernel	**HFMA2.MMA** (100.0%)
KernelUnifiedAttention	**HMMA.16816.F32.BF16** (13.1% - 15.0%), FADD (11.5% - 16.2%), FFMA (4.9% - 7.5%), FMNMX (8.2% - 11.2%), FMUL (42.5% - 56.3%)
PageAttentionV2	**HMMA.16816.F32.BF16** (94.1% - 96.4%), **HMNMX2.BF16_V2** (1.2%), FFMA (2.4%), FMNMX (2.9%), FMUL (2.9%)
RMSNorm	FADD (24.5% - 51.4%), FFMA (18.6% - 25.8%), FMUL (25.7% - 49.1%), MUFU.RCP (0.3% - 2.9%), MUFU.RSQ (0.3% - 1.4%)
ReshapeAndCacheFlash	**HFMA2.MMA** (14.3%), MUFU.RCP (85.7%)
RotaryEmbedding	FADD (31.2%), FMUL (62.5%), MUFU.RCP (6.2%)
SoftmaxReduce	FADD (21.3%), FFMA (17.3%), FMNMX (5.3%), FMUL (45.3%), MUFU.EX2 (6.7%)
SplitKReduce	**HFMA2.MMA** (25.0% - 28.6%), FADD (42.9% - 50.0%), FFMA (12.5% - 14.3%), FMUL (12.5% - 14.3%)
SwiGLU	FADD (6.9%), FFMA (55.2%), FFMA.RM (6.9%), FFMA.SAT (6.9%), MUFU.EX2 (6.9%)
TritonFused1	FADD (55.6% - 75.0%), FFMA (12.5% - 33.3%), FMUL (11.1% - 12.5%)
TritonFused2	**HFMA2.MMA** (1.2%), FFMA (29.6% - 30.0%), FMUL (59.3% - 60.0%), MUFU.RSQ (9.9% - 10.0%)
TritonFused3	FADD (18.2%), FMUL (63.6%), MUFU.EX2 (9.1%), MUFU.RCP (9.1%)
TritonFused4	**HFMA2.MMA** (2.6%), FADD (30.8%), FFMA (12.8%), FMUL (51.3%), MUFU.RSQ (2.6%)
TritonFused5	**HFMA2.MMA** (1.8% - 2.8%), FADD (50.9% - 61.1%), FFMA (6.9% - 9.1%), FMUL (27.8% - 36.4%), MUFU.RSQ (1.4% - 1.8%)

As detailed in Table 1, our analysis reveals a contrasting reality: the vast majority of critical kernels actually perform their internal arithmetic in FP32 to maintain numerical stability. The dominant `GEMM` and `Attention` kernels rely on Tensor Core instructions (*e.g.*, `HMMA.16816.F32.BF16`). Crucially, while these instructions accept BF16 inputs, they inherently utilize `f32` accumulators for the matrix multiplication sums [10]. Operations such as `RotaryEmbedding`, `Softmax`, and `RMSNorm` explicitly cast inputs to FP32 registers before executing standard single-precision arithmetic. We verify this stands true across not only NVIDIA GPUs, but also AMD GPUs.

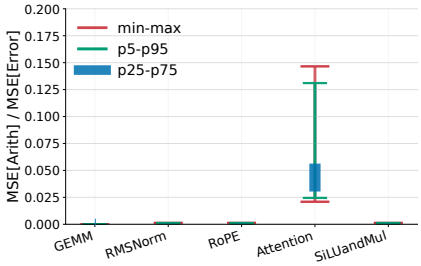


Figure 3: The $\mathbb{E}[\epsilon_{\text{arith}}^2]/\mathbb{E}[(\hat{y}-y)^2]$ ratio of kernels in a vLLM forward pass on NVIDIA A100.

[3]We evaluate this trade-off as the HEAL-Base+Attn+Skip variant in §4.

Because the arithmetic logic itself is nearly identical to a fully stable FP32 pipeline, it cannot be the primary source of numerical error. We can formally verify this by modeling the error of an individual operator $\mathcal{F}$ (*e.g.*, `RotaryEmbedding`, `Softmax`, `GEMM`) at a given layer. Given the input x and the exact output y, suppose we compute under precision p. The overall error is given by:

$$\hat{y} - y = \mathcal{Q}_p\left(\mathcal{F}_p(x)\right) - \mathcal{F}(x) = \underbrace{\left(\mathcal{Q}_p\left(\mathcal{F}_p(x)\right) - \mathcal{F}_p(x)\right)}_{\epsilon_{\text{trunc}}} + \underbrace{\left(\mathcal{F}_p(x) - \mathcal{F}(x)\right)}_{\epsilon_{\text{arith}}} \tag{5}$$

Assuming the values in the tensors have random significands as we observed in our experiments, the quantization error and arithmetic error should be independent. Since modern hardware employs unbiased Round-to-Nearest-Even (RNE) rounding, we derive:

$$\mathbb{E}[\epsilon_{\text{trunc}}] = \mathbb{E}[\epsilon_{\text{arith}}] = 0 \Rightarrow \mathbb{E}[(\hat{y} - y)^2] = \mathbb{E}[\epsilon_{\text{trunc}}^2] + \mathbb{E}[\epsilon_{\text{arith}}^2] \tag{6}$$

Based on this, we can measure the contribution of ϵ_{arith} via the ratio $\mathbb{E}[\epsilon_{\text{arith}}^2]/\mathbb{E}[(\hat{y} - y)^2]$. As shown in Figure 3, we confirm that $\epsilon_{\text{trunc}} \gg \epsilon_{\text{arith}}$ for almost all critical bottleneck kernels, as the arithmetic error is strictly bounded by the FP32 machine epsilon (1.19×10^{-7}) when computation is natively in FP32. The single notable exception is the `Attention` kernel, which exhibits a relatively higher arithmetic error. This anomaly occurs because standard vLLM implementations default to TF32 (TensorFloat-32) within their Triton Attention kernels even under a nominal FP32 pipeline, truncating the machine epsilon 9.77×10^{-4}.

The true driver of instability is exclusively ϵ_{trunc}: the mandatory truncation of high-precision registers down to lower-precision storage at kernel boundaries. To further support our claim, we implemented an entirely FP32 pipeline with manual truncation injected at every kernel boundary. Re-evaluating a subset of MedQA under these controlled conditions, Figure 4 demonstrates that the answer flip rate can be effectively reduced strictly by increasing the number of preserved significand bits. Crucially, we do not need the full 23 significand bits of native FP32 to achieve near-perfect reproducibility; stability is reached much earlier, proving we only need enough fractional precision to cap the noise below the catastrophic divergence threshold.

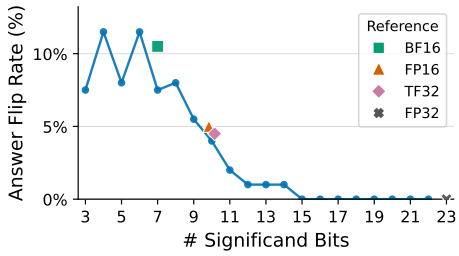


Figure 4: Answer flip rate decreases as significand bits are preserved.

Therefore, we establish the conclusion: non-reproducibility across heterogeneous hardware is not a computation problem; it is strictly a kernel boundary problem. The microscopic seed of all downstream catastrophic divergence is the truncation of the significand during downcasting. Having identified the exact origin of the error, we now introduce our methodology to neutralize this error and secure end-to-end reproducibility.

3 HEAL: Hybrid Error ALleviation

Having established that boundary truncation is the primary driver of numerical divergence, the most straightforward mitigation strategy is a global precision upcast. Specifically, as outlined in the forward pass of Algorithm 1, we could theoretically promote every kernel and intermediate tensor to FP32, eliminating kernel boundary truncation entirely. While this brute-force approach effectively neutralizes the error, it is hardware-inefficient. Intermediate state tensors like $\mathbf{H}$ and $\mathbf{A}$, alongside kernels like `Norm`, can indeed be safely upcast to FP32 with negligible system overhead. However, this global promotion strategy hits a prohibitive performance wall at the network's two most computationally dominant components: the `Attention` and `GEMM` kernels.

For `Attention`, the barrier to upcasting is not computational throughput, but memory capacity. The bottleneck lies entirely within the $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ tensors. Elevating the $\mathbf{K}$ and $\mathbf{V}$ tensors to FP32 strictly doubles the memory footprint of the KV cache, while upcasting $\mathbf{Q}$ significantly inflates intermediate memory usage especially when we have large batches. Because the KV cache already

dictates the memory wall in modern LLM inference, this $2\times$ expansion directly forces a proportional reduction in maximum batch sizes and sequence lengths.

Conversely, for the dense `GEMM` kernel, the barrier shifts to computational throughput. Upcasting these matrix multiplications to FP32 incurs a severe performance penalty that directly contradicts the hardware design of modern AI accelerators. Contemporary architectures are fundamentally optimized for 16-bit arithmetic: NVIDIA's H100 Tensor Cores, for instance, deliver a $14.8\times$ higher throughput for BF16 operations relative to native FP32, while AMD's MI300X Matrix Cores exhibit an $8\times$ advantage. Consequently, a naïve FP32 promotion would severely degrade inference performance.

To achieve both numerical stability and inference efficiency, we introduce **H**ybrid **E**rror **AL**leviation (**HEAL**). HEAL applies a tailored, hardware-aware algebraic strategy to `Attention` and `GEMM`. For `Attention`, HEAL adopts compact INT16 storage and dynamically unpacks it into a high-precision dual-FP16 calculation. For `GEMM`, HEAL maximizes Tensor Core throughput by executing native FP16 matrix multiplications, rescaling the rare outlier channels to avoid overflow and compensating the residual truncation error through a dual-FP16 decomposition. Together, these techniques surgically neutralize the microscopic truncation errors while maintaining the footprint and latency advantages of a native 16-bit pipeline. In practice, HEAL stores K and V with fine-grained group-wise scales to better utilize the INT16 resolution.[4]

Mitigating QKV Truncation Error via INT16 Quantization. To mitigate truncation error within the memory-bound `Attention`, we first observe that standard 16-bit floating-point formats fundamentally underutilize their bit-width when storing $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ tensors. We can formalize this by comparing the expected MSE of the truncation. Given a tensor h with a maximum absolute value M and a standard deviation σ, an INT16 uniform quantization yields an error uniformly distributed between $[-\Delta/2, \Delta/2]$ where $\Delta = \frac{M}{2^{15}-1}$. The expected MSE is therefore:

$$\mathbb{E}[\text{MSE}_{\text{INT16}}] = \frac{\Delta^2}{12} \approx \frac{M^2 \cdot 2^{-30}}{12}. \tag{7}$$

In contrast, for a standard FP16 tensor, assuming a value $x \in h$ falls within the exponent range $2^t < |x| \leq 2^{t+1}$, the truncation error is uniformly distributed within $[-2^{t-12}, 2^{t-12}]$. This yields an expected MSE of:

$$\mathbb{E}[\text{MSE}_{\text{FP16}}] = \mathbb{E}\left[\frac{2^{2t-22}}{12}\right] > \frac{\mathbb{E}[x^2] \cdot 2^{-22}}{12} = \frac{\sigma^2 \cdot 2^{-22}}{12}. \tag{8}$$

This observation dictates that, even in the worst-case scenario, the error ratio is strictly bounded by $\mathbb{E}[\text{MSE}_{\text{FP16}}]/\mathbb{E}[\text{MSE}_{\text{INT16}}] > 256\frac{\sigma^2}{M^2}$. As Figure 5 demonstrates, this translates to a massive empirical precision gain: for 70% of Q, K, V tensors, the theoretical MSE is decreased by at least 50% (and will become 78% when x is uniformly distributed). Driven by this mathematical reality, HEAL discards floating-point storage for the inputs entirely. Instead, any input tensor $\mathbf{X}$ is quantized and stored in INT16: $\mathcal{Q}_{\text{INT16}}(\mathbf{X}) = \text{Round}\left(\frac{(2^{15}-1)\mathbf{X}}{\max|\mathbf{X}|}\right)$. Crucially, this aggressive scaling does not introduce an overflow risk during downstream matrix multiplications. Because the underlying hardware accelerators natively employ FP32 accumulators for the internal summations of GEMV operations, the intermediate scalar products remain safely bounded within high-precision registers.

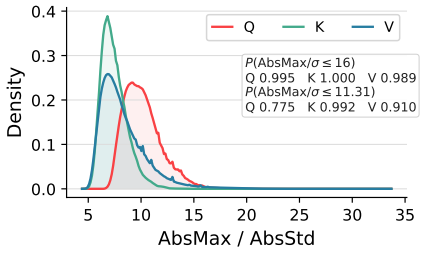


Figure 5: Q, K, V numerical value distribution

To bridge the gap between this optimal storage format and standard hardware—which lacks native Tensor Core support for high-precision INT16 matrix multiplications—HEAL algebraically decomposes the data prior to computation. During execution, the INT16 tensor is unpacked into two FP16 tensors: $\mathbf{X}_{\text{hi}}$ and $\mathbf{X}_{\text{lo}}$. Because a single FP16 value possesses 10 explicit significand bits, shifting the

[4]More implementation details can be found in Appendix C.

lower tensor by a factor of 2^{10} allows the dual-FP16 pairing ($\mathbf{X}_{\text{hi}} + \mathbf{X}_{\text{lo}}/2^{10}$) to seamlessly emulate 20 bits of continuous fractional precision. `Attention` is thus expanded to:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{(\mathbf{Q}_{\text{hi}} + \mathbf{Q}_{\text{lo}}/2^{10})(\mathbf{K}_{\text{hi}} + \mathbf{K}_{\text{lo}}/2^{10})^T}{\sqrt{d_k}}\right)(\mathbf{V}_{\text{hi}} + \mathbf{V}_{\text{lo}}/2^{10}). \tag{9}$$

While this algebraic expansion more than triples the FLOPS, it crucially preserves the strict 16-bit memory footprint. Because the `Attention` is fundamentally memory-bound, this increased computational penalty is effectively absorbed under the roofline model.

Mitigating GEMM Truncation via FP16 Promotion and Algebraic Compensation. To address the remaining input truncation error with `GEMM`, we must balance fractional precision against dynamic range. While standard BF16 prevents overflow, its severely truncated significand drives numerical divergence. Conversely, FP16 natively possesses 10 significand bits—matching the exact fractional precision of TF32. However, this precision comes with an extremely narrow dynamic range, introducing a catastrophic overflow risk: if activation magnitudes exceed the FP16 maximum, the entire inference pass will collapse into `NaN`s.

Fortunately, although there are empirical reports of FP16 overflow [59, 64], such cases occur very infrequently. Throughout our testing process, we found that the maximum absolute value consistently remained below 1024, utilizing less than $\frac{1}{64}$ of the representable FP16 range. Even in the rare cases where overflow might happen, these extreme values only appear in a microscopic fraction (typically 0.1%-1%) of fixed channels [14, 39]. Therefore, HEAL can efficiently handle them by applying targeted rescaling (e.g., weight rescaling) to these exact channels. This allows us to eliminate the overflow risk without adding extra overhead.

Having safely bounded the dynamic range, HEAL shifts all standard `GEMM` operations to FP16 to harvest its TF32-equivalent precision. Furthermore, when strict task reproducibility demands precision beyond a single FP16 representation, HEAL applies the same dual-FP16 pairing strategy introduced in `Attention`. We mathematically decompose the high-precision FP32 input tensor h as $(h_{\text{hi}} + h_{\text{lo}}/2^{10})$. The original dense computation is thus expanded into two parallel, high-throughput FP16 operations:

$$\text{GEMM}(h, \mathbf{W}^T) \approx \text{GEMM}(h_{\text{hi}}, \mathbf{W}^T) + \text{GEMM}(h_{\text{lo}}/2^{10}, \mathbf{W}^T). \tag{10}$$

Similar to our approach in `Attention`, computing the `GEMM` with the h_{lo} tensor seamlessly injects additional fractional bits into the final output. This allows HEAL to approximate FP32 precision while keeping all computations fully optimized for standard Tensor Cores.

4 Evaluation

Evaluation Scope and Metrics. For clarity, we distinguish three closely related but conceptually distinct notions: *(i) Cross-hardware reproducibility*, the requirement that a fixed model and input yield functionally identical downstream task outcomes across heterogeneous hardware configurations; *(ii) Agreement with FP32*, the degree to which low-precision execution aligns with a stable H100 FP32 reference anchor; and *(iii) Task accuracy*, whether an output matches the dataset's objective ground-truth label. Crucially, task accuracy reflects a model's intrinsic capability and is orthogonal to reproducibility: HEAL does not aim to alter model accuracy, but rather prevents boundary truncation noise from flipping the model's deterministic prediction across devices.

To rigorously evaluate HEAL under real-world conditions, we introduce the Mission-Critical Reproducibility Benchmark (MCR-Bench). MCR-Bench curates 4500 high-stakes reasoning problems from the medical, legal, and financial domains, specifically targeting complex scenarios where numerical divergence translates into catastrophic real-world failures.[5] To quantify reproducibility, our primary metric is the answer flip rate, which measures the percentage of generated responses that diverge from the FP32 anchor. Alongside this, we report the Final Hidden MSE, which, as established earlier, captures the mean squared error in the normalized representations immediately before the vocabulary projection matrix. Finally, to evaluate system efficiency, we utilize the ShareGPT dataset [1] to measure both the Time Per Output Token (TPOT) and the base memory footprint (accounting strictly

[5] More details about MCR-Bench can be found in Appendix E.

for model weights and computational workspace). These performance metrics demonstrate that HEAL successfully circumvents the severe computational slowdowns associated with full-precision pipelines, achieving high reproducibility while maintaining minimal performance overhead.

Setup & Baselines. We evaluate our system across three diverse hardware types: NVIDIA A100, NVIDIA H100, and AMD MI300X. We compare against four baseline precisions: BF16, FP16, TF32, and FP32 with LayerCast [70] applied for 32-bit baselines. Alongside our complete system (HEAL), we evaluate three cumulative ablation steps (HEAL-Base, HEAL-Base+Attn, and HEAL-Base+Attn+Skip) to isolate the contribution of each component: i) HEAL-Base: maintains the residual stream in FP32 while promoting standard `GEMM` layers to FP16; ii) HEAL-Base+Attn: integrates INT16 quantization and dual-FP16 unpacking for the `Attention` QKV tensors; iii) HEAL-Base+Attn+Skip: selectively bypasses upcasting for early layers whose error contributions safely remain below the result flipping threshold; iv) HEAL: the complete recommended pipeline, incorporating the dual-FP16 pairing compensation for `GEMM` to maximize fractional precision.[6]

Reproducibility. We first evaluate end-to-end reproducibility on MCR-Bench. As we observe that the accumulated numerical error remains remarkably consistent across hardware platforms under a fixed precision configuration, we report the metrics averaged across all three GPUs in Table 2. Note that HEAL does not systematically change task accuracy.[7]

Table 2: Final Hidden MSE and answer flip rate across five models on MCR-Bench.

Approach	Qwen3-8B		Qwen3-14B		Qwen3-32B		Llama-8B		DeepSeek-14B	
	MSE	Flip Rate	MSE	Flip Rate	MSE	Flip Rate	MSE	Flip Rate	MSE	Flip Rate
BF16	1.6e-3	9.70%	1.3e-3	8.03%	4.6e-3	9.18%	1.9e-3	16.53%	1.4e-3	13.90%
FP16	6.9e-5	4.90%	2.6e-5	3.84%	8.5e-5	4.50%	3.9e-5	3.82%	3.3e-5	6.03%
TF32 (+LayerCast)	2.2e-5	2.87%	1.2e-5	2.11%	4.9e-5	2.47%	2.8e-5	1.97%	1.6e-5	3.77%
FP32 (+LayerCast)	1.9e-10	0.01%	1.6e-10	0.01%	6.9e-10	0.00%	3.2e-10	0.01%	1.4e-10	0.01%
HEAL-Base	9.9e-6	2.16%	6.9e-6	1.36%	1.4e-5	1.73%	1.1e-5	1.22%	5.0e-6	2.39%
HEAL-Base+Attn	3.4e-6	1.25%	1.4e-6	0.92%	4.3e-6	1.02%	2.8e-6	0.66%	1.6e-6	1.25%
HEAL-Base+Attn+Skip	4.3e-6	1.48%	5.5e-6	1.01%	5.4e-6	1.09%	7.3e-6	0.84%	2.9e-6	1.34%
HEAL	6.5e-8	0.06%	1.3e-8	0.07%	5.4e-8	0.07%	3.5e-8	0.10%	1.0e-8	0.07%

* *Llama-8B* refers to Llama-3.1-8B-Instruct and *DeepSeek-14B* refers to DeepSeek-Distill-Qwen-14B.

As demonstrated in Table 2, native low-precision executions (BF16 and FP16) exhibit high token divergence due to boundary truncation. HEAL-Base successfully pulls the accumulated MSE back into the benign regime. Ultimately, HEAL neutralizes microscopic errors, closely aligned with the flip rate and MSE of the FP32 baseline, justifying that reproducibility does not require a global high-precision pipeline.

Performance Overhead. To benchmark practical deployment overhead, we evaluate TPOT and memory overhead by processing real-world inference workloads from the ShareGPT [1] dataset. For brevity and representativeness, we only present the results on Qwen3-14B in Table 3. The results for the remaining models are provided in Appendix D.

Table 3: ShareGPT TPOT and Memory Usage for the **Qwen3-14B** model.

Configuration	A100 TPOT (ms)	H100 TPOT (ms)	MI300X TPOT (ms)	Memory Usage (GiB)
BF16	108.2 ± 0.3	41.7 ± 0.1	55.4 ± 0.2	28.1
TF32 (+LayerCast)	259.3 ± 1.9	114.8 ± 0.1	128.1 ± 0.2	29.6
FP32 (+LayerCast)	1332.3 ± 5.9	546.5 ± 1.2	243.2 ± 0.4	29.6
HEAL-Base	120.5 ± 0.1	47.5 ± 0.0	59.6 ± 0.1	28.3
HEAL-Base+Attn	130.3 ± 0.8	53.7 ± 0.1	78.0 ± 0.2	28.3
HEAL-Base+Attn+Skip	126.4 ± 0.7	51.6 ± 0.2	71.9 ± 0.2	28.3
HEAL	235.5 ± 1.4	91.7 ± 0.5	116.9 ± 0.2	28.5

While the FP32 baseline severely degrades throughput by bypassing Tensor Cores, our progressive variants (HEAL-Base and HEAL-Base+Attn) preserve a strict 16-bit KV-cache footprint with modest latency costs, and HEAL-Base+Attn+Skip further recaptures throughput by skipping redundant

[6] More details about the experiment setup and results can be found in Appendix A.
[7] More details about this can be found in Appendix B.6.

early-layer upcasts. Across all platforms on Qwen3-14B, the complete HEAL pipeline operates within the native 16-bit memory footprint while incurring an average TPOT overhead of 116.2% relative to BF16, avoiding the prohibitive slowdowns of full FP32 execution to deliver practical, mission-critical reproducibility.[8]

5 Related Work

Reproducibility in LLM inference. Achieving bitwise-identical inference is challenging due to numerical non-associativity and system-level variance (*e.g.*, parallel reductions) [70]. Existing mitigation strategies typically attempt to enforce strict bit-level consistency by constraining execution paths, such as employing framework-level deterministic modes, batch-invariant kernels [26, 60], or tensor-parallel alignment [72]. While these techniques mitigate run-to-run variance within a fixed software-hardware stack, they fundamentally fail across heterogeneous GPUs with disparate instruction sets and reduction implementations. Other approaches, such as LLM-42 [20] with verified speculation and ReproBLAS [13] with order-independent specialized accumulators at the operator level, would require a deterministic pipeline, which is exceedingly difficult to guarantee across heterogeneous hardware, and often incur substantial performance overhead. In contrast, instead of enforcing bitwise determinism, HEAL suppresses kernel-boundary truncation noise well below the argmax flipping threshold, achieving cross-hardware reproducibility while directly reusing native hardware kernels.

Mixed precision in LLM inference. Mixed-precision strategies improve throughput by performing sensitive operations (*e.g.*, attention, accumulation) in FP32 while storing data in BF16/FP16 [10, 37, 56]. Recent work also advocates for native FP16 execution [51] to eliminate training-inference mismatches, but lack deep investigation into fundamental instability mechanisms (*e.g.*, kernel-boundary truncation) and offer no systematic mitigation against catastrophic overflow. Moreover, reproducibility still fails due to precision loss when writing back to lower precision and hardware-specific IEEE-754 behaviors [9], which induce consistent but device-dependent perturbations.

Quantization and numerical sensitivity. Existing work on post-training quantization highlights how outliers dominate error, requiring specific smoothing or scaling strategies to preserve quality [14]. While this line of work targets the accuracy-throughput trade-off rather than reproducibility, its findings align with our analysis of how rounding effects amplify drift. Specifically, weight-activation [14, 66] and low-bit KV-cache quantization [28, 40] inherently downcast intermediate states at kernel boundaries, introducing additional truncation that further degrades reproducibility. In contrast, weight-only quantization [16, 39] compresses static parameters without introducing additional runtime truncation, making it orthogonal to and composable with HEAL (Appendix B.5). HEAL itself builds upon established numerical primitives, such as multi-component expansion [12, 45] and channel-wise scaling [39, 66]; HEAL's contribution lies in the diagnosis of kernel-boundary truncation and its application to cross-hardware reproducibility.

6 Conclusion

This work demonstrates that cross-hardware non-reproducibility in LLM inference stems not from internal kernel arithmetic, but from significand truncation during downcasting at kernel boundaries, which accumulates across layers to trigger answer flips. To avoid the prohibitive cost of global FP32 execution, we propose HEAL, a hybrid framework that bounds boundary truncation errors below the result-flipping threshold via compact INT16 Q/K/V quantization and dual-FP16 algebraic GEMM compensation on native 16-bit Tensor Cores. Evaluated on MCR-Bench across diverse models and accelerators, HEAL achieves FP32-level functional reproducibility within the native 16-bit memory footprint while reducing the TPOT overhead of the FP32 pipeline by up to 6.0×.

Acknowledgments

We thank the anonymous reviewers for their comments. This work was supported by the National Science Foundation (CNS-2301343, CNS-2330831, CNS-2403254, IIS-2546642, IIS-2551122), as well as the Alibaba Research Intern Program. Junyi Shu is the corresponding author.

[8] More details about this can be found in Appendix D.

References

[1] anon8231489123. ShareGPT vicuna unfiltered dataset. https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered, 2023.

[2] Anthropic. Powering the next generation of AI development with AWS. https://www.anthropic.com/news/anthropic-amazon-trainium, 2024.

[3] Anthropic. Anthropic expands partnership with Google and Broadcom for multiple gigawatts of next-generation compute. https://www.anthropic.com/news/google-broadcom-partnership-compute, 2026.

[4] Anna Arias-Duart, Pablo Agustin Martin-Torres, Daniel Hinjos, Pablo Bernabeu-Perez, Lucia Urcelay Ganzabal, Marta Gonzalez Mallo, Ashwin Kumar Gururajan, Enrique Lopez-Cuena, Sergio Alvarez-Napagao, and Dario Garcia-Gasulla. Automatic evaluation of healthcare llms beyond question-answering. 2025. URL https://arxiv.org/pdf/2502.06666.

[5] Andrew Blair-Stanek and Benjamin Van Durme. LLMs Provide Unstable Answers to Legal Questions, 2025. URL https://arxiv.org/abs/2502.05196.

[6] Nicholas C Cardamone, Mark Olfson, Timothy Schmutte, Lyle Ungar, Tony Liu, Sara W Cullen, Nathaniel J Williams, and Steven C Marcus. Classifying Unstructured Text in Electronic Health Records for Mental Health Prediction Models: Large Language Model Evaluation Study. *JMIR Medical Informatics*, 13(1):e65454, 2025.

[7] Ilias Chalkidis, Abhik Jana, Dirk Hartung, Michael Bommarito, Ion Androutsopoulos, Daniel Katz, and Nikolaos Aletras. LexGLUE: A benchmark dataset for legal language understanding in English. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4310–4330, Dublin, Ireland, May 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.acl-long.297.

[8] Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. Finqa: A dataset of numerical reasoning over financial data. *Proceedings of EMNLP 2021*, 2021.

[9] NVIDIA Corporation. *Floating Point and IEEE 754 Compliance for NVIDIA GPUs*. URL https://docs.nvidia.com/cuda/floating-point/index.html. CUDA Toolkit Documentation.

[10] NVIDIA Corporation. NVIDIA A100 Tensor Core GPU Architecture. Whitepaper, NVIDIA, 2020. URL https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf.

[11] NVIDIA Corporation. Using the cuBLASLt API. https://docs.nvidia.com/cuda/cublas/#using-the-cublaslt-api, 2026.

[12] Theodorus J. Dekker. A floating-point technique for extending the available precision. *Numerische Mathematik*, 18(3):224–242, 1971. doi: 10.1007/BF01397083.

[13] James Demmel, Peter Ahrens, and Hong Diep Nguyen. Efficient reproducible floating point summation and blas. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-121*, 2016.

[14] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale. *Advances in neural information processing systems*, 35:30318–30332, 2022.

[15] Centers for Medicare & Medicaid Services. ICD Code Lists. https://www.cms.gov/medicare/coordination-benefits-recovery/overview/icd-code-lists, 2026.

[16] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations (ICLR)*, 2023.

[17] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The Language Model Evaluation Harness. https://zenodo.org/records/12608602, 2024.

[18] Aryo Pradipta Gema, Joshua Ong Jun Leang, Giwon Hong, Alessio Devoto, Alberto Carlo Maria Mancino, Rohit Saxena, Xuanli He, Yu Zhao, Xiaotang Du, Mohammad Reza Ghasemi Madani, Claire Barale, Robert McHardy, Joshua Harris, Jean Kaddour, Emile van Krieken, and Pasquale Minervini. Are we done with MMLU? In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 2025.

[19] Nayoon Gim, In Gim, Yu Jiang, Yuka Kihara, Marian Blazes, Yue Wu, Cecilia S Lee, and Aaron Y Lee. An LLM-assisted framework for accelerated and verifiable clinical hypothesis testing from electronic health records. *medRxiv*, 2026.

[20] Raja Gond, Aditya K Kamath, Ramachandran Ramjee, and Ashish Panwar. Llm-42: Enabling determinism in llm inference with verified speculation. *arXiv preprint arXiv:2601.17768*, 2026.

[21] Aleksandr Gornostal, Lori Mazor, and Rachel Curtis. 6 ways generative AI is transforming financial services. https://star.global/posts/generative-ai-in-finance/.

[22] 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.

[23] Neel Guha, Julian Nyarko, Daniel E. Ho, Christopher Ré, Adam Chilton, Aditya Narayana, Alex Chohlas-Wood, Austin Peters, Brandon Waldon, Daniel N. Rockmore, Diego Zambrano, Dmitry Talisman, Enam Hoque, Faiz Surani, Frank Fagan, Galit Sarfaty, Gregory M. Dickinson, Haggai Porat, Jason Hegland, Jessica Wu, Joe Nudell, Joel Niklaus, John Nay, Jonathan H. Choi, Kevin Tobia, Margaret Hagan, Megan Ma, Michael Livermore, Nikon Rasumov-Rahe, Nils Holzenberger, Noam Kolt, Peter Henderson, Sean Rehaag, Sharad Goel, Shang Gao, Spencer Williams, Sunny Gandhi, Tom Zur, Varun Iyer, and Zehua Li. Legalbench: A collaboratively built benchmark for measuring legal reasoning in large language models, 2023.

[24] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948*, 2025.

[25] Nathan Habib, Clémentine Fourrier, Hynek Kydlíček, Thomas Wolf, and Lewis Tunstall. LightEval: A Lightweight Framework for LLM Evaluation. https://github.com/huggingface/lighteval, 2023.

[26] Horace He and Thinking Machines Lab. Defeating Nondeterminism in LLM Inference. https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/, September 2025. Accessed: 2026-01-19.

[27] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring Massive Multitask Language Understanding. In *International Conference on Learning Representations (ICLR)*, 2021.

[28] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, 2024.

[29] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling (COLM)*, 2024.

[30] Advanced Micro Devices Inc. hipBLASLt documentation. https://rocm.docs.amd.com/projects/hipBLASLt/en/latest/, 2026.

[31] Ingonyama. Solving Reproducibility Challenges in Deep Learning and LLMs: Our Journey. https://www.ingonyama.com/post/solving-reproducibility-challenges-in-deep-learning-and-llms-our-journey, September 2024.

[32] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations (ICLR)*, 2025.

[33] Rounak Jain. Nvidia Says Its 6-Year Old A100 Chips Are Running At Full Tilt, Countering Michael Burry's Depreciation Warning. https://stocktwits.com/news-articles/markets/equity/nvidia-a100-gpu-shipped-6-years-ago-full-utilization-michael-burry-depreciation/cLPALD9RE9w.

[34] Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021.

[35] Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William Cohen, and Xinghua Lu. Pubmedqa: A dataset for biomedical research question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2567–2577, 2019.

[36] Raffi Khatchadourian and Rolando Franco. LLM Output Drift: Cross-Provider Validation & Mitigation for Financial Workflows, 2025. URL https://arxiv.org/abs/2511.07585.

[37] 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.

[38] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2024.

[39] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6:87–100, 2024.

[40] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *PMLR*, pages 32332–32344, 2024.

[41] Macedo Maia, Siegfried Handschuh, André Freitas, Brian Davis, Ross McDermott, Manel Zarrouk, and Alexandra Balahur. WWW'18 Open Challenge: Financial Opinion Mining and Question Answering. In *Companion Proceedings of The Web Conference 2018*, pages 1941–1942, 2018. doi: 10.1145/3184558.3192301.

[42] Pekka Malo, Ankur Sinha, Pekka Korhonen, Jyrki Wallenius, and Pyry Takala. Good Debt or Bad Debt: Detecting Semantic Orientations in Economic Texts. *Journal of the Association for Information Science and Technology*, 65(4):782–796, 2014. doi: 10.1002/asi.23062.

[43] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer Sentinel Mixture Models, 2016.

[44] Qais Noorshams, Chinmaya Mishra, Harald Seipp, Sailendu Patra, Dietmar Fischer, Kedar Karmarkar, and Mathias Defiebre. Next-generation Enterprise AI solutions with IBM watsonx.ai and IBM Storage Scale. https://www.redbooks.ibm.com/redpieces/pdfs/redp5765.pdf, 2026.

[45] Hiroyuki Ootomo and Rio Yokota. Recovering single precision accuracy from Tensor Cores while surpassing the FP32 theoretical peak performance. *The International Journal of High Performance Computing Applications*, 36(4):475–491, 2022. doi: 10.1177/10943420221090256.

[46] OpenAI. Multilingual MMLU (MMMLU) dataset. https://huggingface.co/datasets/openai/MMMLU, 2024.

[47] OpenAI. AMD and OpenAI announce strategic partnership to deploy 6 gigawatts of AMD GPUs. https://openai.com/index/openai-amd-strategic-partnership/, 2025.

[48] OpenAI. OpenAI and Amazon announce strategic partnership. https://openai.com/index/amazon-partnership/, 2026.

[49] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[50] PyTorch Contributors. Reproducibility — PyTorch documentation. https://docs.pytorch.org/docs/stable/notes/randomness.html.

[51] Penghui Qi, Zichen Liu, Xiangxin Zhou, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Defeating the training-inference mismatch via fp16. *arXiv preprint arXiv:2510.26788*, 2025.

[52] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv e-prints*, 2019.

[53] Burkhard Ringlein, Jan van Lunteren, Radu Stoica, and Thomas Parnell. The Anatomy of a Triton Attention Kernel, 2025. URL https://arxiv.org/abs/2511.11581.

[54] Jérémie Sanfaçon. When temperature=0 isn't zero: A Bedrock Determinism Bug in LangChain AWS. https://www.coveo.com/blog/bedrock-determinism-bug-in-langchain-aws/, 2026.

[55] Pranab Kumar Sen. Estimates of the regression coefficient based on kendall's tau. *Journal of the American statistical association*, 63(324):1379–1389, 1968.

[56] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-precision. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, pages 68658–68685, 2024.

[57] sksayan01. Finclaimradar: A dataset for evidentiary reasoning and classification of financial claims, 2025. URL https://huggingface.co/datasets/sksayan01/FinClaimRadar-Financial-Reasoning.

[58] Da Song, Yuheng Huang, Boqi Chen, Tianshuo Cong, Randy Goebel, Lei Ma, and Foutse Khomh. Evaluating Implicit Regulatory Compliance in LLM Tool Invocation via Logic-Guided Synthesis, 2026. URL https://arxiv.org/abs/2601.08196.

[59] Mingjie Sun, Xinlei Chen, J Zico Kolter, and Zhuang Liu. Massive activations in large language models. *arXiv preprint arXiv:2402.17762*, 2024.

[60] The SGLang Team. Towards deterministic inference in sglang and reproducible rl training. https://lmsys.org/blog/2025-09-22-sglang-deterministic/, 2025.

[61] Philippe Tillet, H. T. Kung, and David Cox. Triton: An intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19, 2019. doi: 10.1145/3315508.3329973.

[62] Charlotte Trueman. AWS has "never retired" an Nvidia A100 server, CEO Matt Garman claims. https://www.datacenterdynamics.com/en/news/aws-has-never-retired-an-nvidia-a100-server-ceo-matt-garman-claims/, 2026.

[63] Ehsan Ullah, Anil Parwani, Mirza Mansoor Baig, and Rajendra Singh. Challenges and barriers of using large language models (LLM) such as ChatGPT for diagnostic medicine with a focus on digital pathology – a recent scoping review. *Diagnostic pathology*, 19(1), 2024. ISSN 1746-1596.

[64] wenqiangire-commits. [bug]: Gemma 4 (31b/26b-a4b) vision outputs only <pad> under fp16 — vision_tower standardize overflows. GitHub Issue #40290, 2026. URL `https://github.com/vllm-project/vllm/issues/40290`.

[65] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[66] 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 *International conference on machine learning*, pages 38087–38099. PMLR, 2023.

[67] Jia Xu, Weilin Du, Xiao Liu, and Xuejun Li. Llm4workflow: An llm-based automated workflow model generation tool. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pages 2394–2398, 2024.

[68] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*, 2025.

[69] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, 2018. doi: 10.18653/v1/D18-1259.

[70] Jiayi Yuan, Hao Li, Xinheng Ding, Wenya Xie, Yu-Jhe Li, Wentian Zhao, Kun Wan, Jing Shi, Xia Hu, and Zirui Liu. Understanding and mitigating numerical sources of nondeterminism in llm inference. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[71] Shengbin Yue, Shujun Liu, Yuxuan Zhou, Chenchen Shen, Siyuan Wang, Yao Xiao, Bingxuan Li, Yun Song, Xiaoyu Shen, Wei Chen, Xuanjing Huang, and Zhongyu Wei. LawLLM: Intelligent Legal System with Legal Reasoning and Verifiable Retrieval. In *Lecture Notes in Computer Science*, volume 14854 of *Database Systems for Advanced Applications*. Springer-Verlag, 2024. ISBN 978-981-97-5568-4.

[72] Ziyang Zhang, Xinheng Ding, Jiayi Yuan, Rixin Liu, Huizi Mao, Jiarong Xing, and Zirui Liu. Deterministic Inference across Tensor Parallel Sizes That Eliminates Training-Inference Mismatch, 2025. URL `https://arxiv.org/abs/2511.17826`.

[73] Lucia Zheng, Neel Guha, Brandon R Anderson, Peter Henderson, and Daniel E Ho. When does pretraining help? assessing self-supervised learning for law and the casehold dataset of 53,000+ legal holdings. In *Proceedings of the eighteenth international conference on artificial intelligence and law*, pages 159–168, 2021.

[74] Lucia Zheng, Neel Guha, Javokhir Arifov, Sarah Zhang, Michal Skreta, Christopher D Manning, Peter Henderson, and Daniel E Ho. A reasoning-focused legal retrieval benchmark. In *Proceedings of the 2025 Symposium on Computer Science and Law*, pages 169–193, 2025.

[75] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-Following Evaluation for Large Language Models, 2023. URL `https://arxiv.org/abs/2311.07911`.

A Detailed Experimental Setup

Setup. We conducted our experiments on top of vLLM [37], a commonly used inference engine. To ensure reproducibility and align with prior work [70], we strictly controlled the software environment. Specifically, the test environment was initialized using an official Ubuntu 22.04 Docker base image with g++-11, Python 3.12.13, and vLLM v0.19.0 installed. We use CUDA 12.8 with the standard Release 580 series drivers for NVIDIA GPUs, and ROCm 7.2.1 for AMD GPUs.

To ensure our findings represent fundamental systemic behaviors rather than artifacts of a specific architecture, our diagnostic analysis spans five widely deployed open-source models: Qwen3-8B, 14B, and 32B [68], Llama-3.1-8B-Instruct [22], and DeepSeek-R1-Distill-Qwen-14B [24]. We used one A100 80GB SXM4 GPU, one H100 80GB SXM5 GPU, and one AMD MI300X GPU to conduct the experiments. There is no GPU sharing when serving the models, and the temperature is fixed to zero unless otherwise specified.

Baselines. Throughout our experiments, we evaluate four baselines with varied precision formats: BF16, FP16, TF32, and FP32. Specifically, these formats feature varying numbers of significand bits for precision: 7 for BF16, 10 for both FP16 and TF32, and 23 for FP32. For the 32-bit baselines (TF32 and FP32), we execute them with LayerCast [70] enabled. While native FP32 and FP32 with LayerCast are theoretically equivalent, exhibiting only minor variations in their compiled CUDA graphs, we opt for LayerCast due to its reduced memory overhead. This design choice is practically necessary, as it enables us to serve Qwen3-32B entirely in 32-bit precision on a single 80GB GPU.

To isolate the impact of numerical representation and ensure a fair comparison across all precisions, we implement all baselines using TritonAttention, since standard FlashAttention currently lacks support for 32-bit formats. Crucially, to ensure rigorous 32-bit execution for the FP32 baseline, we explicitly set the `input_precision` argument to `"ieee"` for all `tl.dot` operations within the TritonAttention kernel. Without this explicit enforcement, the core matrix multiplications inside the TritonAttention silently fall back to TF32, introducing unintended numerical noise that confounds exact reproducibility analysis.

Experiments. Throughout our evaluations, we use the FP32 LayerCast execution on the NVIDIA H100 as our reference anchor. Unless otherwise specified, we use vLLM's default parameters. Note that CUDA Graphs are disabled during our numerical and reproducibility analyses, and are exclusively enabled when measuring end-to-end inference performance.

Datasets. For the motivating experiments in §2 and §3, we randomly sample 200 questions from the 1273 questions in the MedQA benchmark [34]. To elicit detailed reasoning while preventing unbounded auto-regressive outputs, we enable the models' Chain-of-Thought (CoT) capabilities and enforce a conservative generation cap of 4096 tokens. We append a strict formatting instruction to the default prompt template to facilitate automated parsing:

```
"You are solving a US medical licensing multiple-choice question. Think step by step,
then end with exactly one final line formatted as 'Final Answer: X' where X is A, B, C,
or D."
```

Table 4 summarizes the execution statistics of this MedQA subset, utilizing the H100 FP32 LayerCast execution as our reference. The "Truncated Due to Length" metric reflects the rare instances where a model hit the aforementioned token cap, while "Successfully Parsed" indicates outputs that strictly adhered to our requested final answer format.

Table 4: H100 FP32 LayerCast Reference Statistics for the MedQA-200 Subset

Metric	Qwen3-8B	Qwen3-14B	Qwen3-32B	Llama-8B	DS-14B
# Questions	200	200	200	200	200
Median Response Length	1022.5	751.5	823.5	353	1421
Truncated Due to Length	9/200	2/200	5/200	4/200	12/200
Successfully Parsed	191/200	198/200	193/200	195/200	188/200
# Generated Tokens	295616	213541	222327	86354	337479
Accuracy	149/200	161/200	175/200	137/200	144/200

* *Llama-8B* refers to Llama-3.1-8B-Instruct and *DS-14B* refers to DeepSeek-R1-Distill-Qwen-14B.

For the MSE error and Flip Rate Analysis in §4, we transition to **MCR-Bench**, an evaluation protocol curated from existing datasets and specifically designed to assess LLM reproducibility on downstream tasks. MCR-Bench encompasses diverse workloads and manages its own dynamic generation constraints (see Appendix E for further details).

For the performance evaluation in §4, we use the ShareGPT V3 dataset [1] under the following vLLM serving setup:

```
vllm serve <model> \
  --max-model-len 8192 \
  --max-num-batched-tokens 8192 \
  --max-num-seqs 256 \
  --gpu-memory-utilization 0.90

vllm bench serve \
  --dataset-name sharegpt \
  --num-prompts 1024 \
  --request-rate inf \
  --seed 0 \
  --temperature 0
```

The detailed execution time of experiments is summarized in Table 5. In total, the reported experiments take approximately 440 A100 hours, 275 H100 hours, and 150 MI300X hours.

Table 5: Execution Time of Each Experiment (GPU Hours)

Task	**GPU**	**Qwen3-8B**	**Qwen3-14B**	**Qwen3-32B**	**Llama-8B**	**DS-14B**
Reference Anchor Generation	H100	0.2	0.2	1.9	0.1	0.4
Error Contribution (Figure 2, 8)	H100	0.4	0.6	1.1	0.2	0.6
Final Hidden MSE Analysis (Figure 1, 6)	A100	9.1	8.6	15.7	3.1	12.9
	H100	4.5	4.2	8.3	1.6	6.4
	MI300X	3.2	3.3	6.3	1.3	5.0
Flip Rate Analysis (Figure 1, 4, 6, 7)	MI300X	1.8	2.2	6.2	1.2	2.8
Arithmetic vs Truncation (Figure 3, 9)	A100	0.1	0.2	1.9	0.1	0.2
QKV Distribution (Figure 5, 10)	H100	0.1	0.1	0.6	0.1	0.1
Weight-only Quantization (Table 6)	A100	7.6	7.9	25.2	3.0	12.0
MCR-Bench (Table 2)	A100	26.9	45.7	167.9	18.2	57.0
	H100	26.3	39.2	113.0	17.2	41.0
	MI300X	15.6	21.2	46.1	8.8	22.1
ShareGPT V3 (Table 3, 10, 11, 12, 13)	A100	2.0	2.9	8.8	1.9	3.0
	H100	0.5	0.8	3.4	0.5	0.9
	MI300X	0.8	1.1	2.1	0.7	1.1

* *Llama-8B* refers to Llama-3.1-8B-Instruct and *DS-14B* refers to DeepSeek-R1-Distill-Qwen-14B.

Evaluation Metrics and Error Bars. For all experiments where error bars are reported, we display the mean alongside the standard error of the mean (SEM). Formally, this interval is represented as:

$$\mu \pm \frac{\sigma}{\sqrt{N}} \tag{11}$$

where μ is the sample mean, σ is the standard deviation, and N represents the sample size (e.g., the total number of evaluated tokens for MSE evaluations, or the number of independent runs for generation experiments). For answer flip rates, the dominant source of variability is the finite question sample. With 4500 questions per GPU, the binomial standard error $\sqrt{p(1-p)/N}$ is at most 0.6% for the baselines and about 0.05% for HEAL.

B Additional Study Results

This section presents the results of our study on the set of supplementary models, including Qwen3-8B, Llama3.1-8B-Instruct (referred as **Llama-8B**), DeepSeek-R1-Distill-Qwen-14B (referred as **DeepSeek-14B**, **DS-14B**), and Qwen3-32B.

B.1 Error and Flip Rate (Figure 1 and Figure 4)

To investigate the relationship between numerical precision and model behavior, we developed a two-stage evaluation pipeline. We first implement an FP32 execution path with manual significand truncation injected at every kernel boundary. This pipeline is controlled by a parameter k, representing the number of preserved significand bits. By varying k from 3 to 22 (when $k = 23$ it becomes FP32), we generate 20 distinct precision modes. Combined with our 4 baselines, this results in 24 distinct configurations to evaluate. Our diagnostic analysis focuses on two primary metrics: Final Hidden Layer MSE and Answer Flip Rate.

Final Hidden Layer MSE Analysis. To isolate numerical divergence from the compounding effects of different token histories, we utilize Teacher Forcing (replaying the reference anchor token trajectory). We use the FP32 LayerCast results on the NVIDIA H100 as the anchor. At each step, to quantify the accumulated noise just before the final decision, we measure the Final Hidden MSE, defined as the mean squared error in the representations immediately before the vocabulary projection matrix (i.e., the normalized final hidden layer output). We deliberately evaluate on this hidden dimension rather than the pre-softmax logits because its significantly smaller size allows us to substantially scale up our evaluation data volume.

Moreover, the pure replay mechanism avoids auto-regressive generation, which significantly accelerates the data collection process. Consequently, we are able to evaluate the MSE across all three hardware platforms: NVIDIA A100, NVIDIA H100, and AMD MI300X.

Answer Flip Rate Analysis. To evaluate how internal numerical errors manifest as actual token deviations, we calculate the Answer Flip Rate, which requires a full auto-regressive decoding process. Based on our prompt format, we parse the model's final response into one of five discrete categories: A, B, C, D, or F, where F denotes a failure to successfully parse a formatted answer. An "Answer Flip" is explicitly defined as any instance where this categorical outcome diverges from our anchor execution (H100 FP32 LayerCast).

Executing the 20 precision-controlled modes requires falling back to the FP32 execution path to apply manual truncation, which imposes a heavy demand on the hardware's FP32 throughput. Furthermore, auto-regressive generation inherently suffers from poor concurrency. To achieve acceptable hardware utilization and generation efficiency, a large batch size is strictly required. However, the 80GB memory capacity of the A100 and H100 GPUs severely limits the maximum batch size, making efficient generation impossible and resulting in prohibitively high execution times under these constraints. Consequently, we conduct the Flip Rate analysis exclusively on the AMD MI300X. Its massive 192GB memory capacity comfortably accommodates the large batch sizes needed to overcome the low concurrency of generation, while its superior FP32 throughput (163.4 TFLOPS, compared to 67 TFLOPS on H100 and 19.5 TFLOPS on A100) effectively absorbs the heavy overhead of manual truncation.

Figure 1 and Figure 6 correlate these two metrics to visualize how incremental precision loss at the kernel level eventually manifests as discrete token divergence. Notably, the empirical results indicate that under a fixed precision configuration, the final hidden MSE remains hardware-agnostic. Figure 4 and Figure 7 further isolate the impact of significand bit width on the stability of the model output. Table 5 summarizes the execution time required for both analysis pipelines.

Observations on High-Precision Flips. Through manual inspection, we observed that a substantial portion of the answer flips, especially those occurring under relatively high-precision configurations, generally originate from highly unstable questions. For these specific questions, the model typically exhibits severe uncertainty in its reasoning trace, often debating between multiple choices or doubting the validity of all options. This extreme hesitation can easily derail the auto-regressive generation, sometimes causing the model to degenerate into a self-repeating loop or simply fail to reach a definitive conclusion, ultimately leading to a parse failure (F). Under such inherently unstable conditions, even microscopic numerical perturbations can easily alter the decoding trajectory. Consequently, while

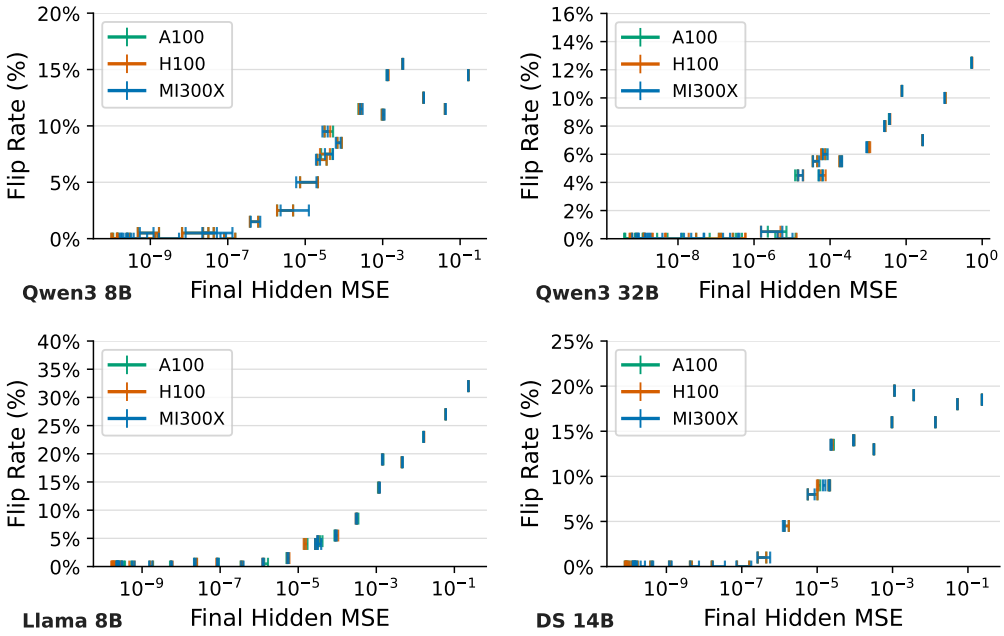


Figure 6: Answer flip rate vs. total numerical error, complement of Figure 1.

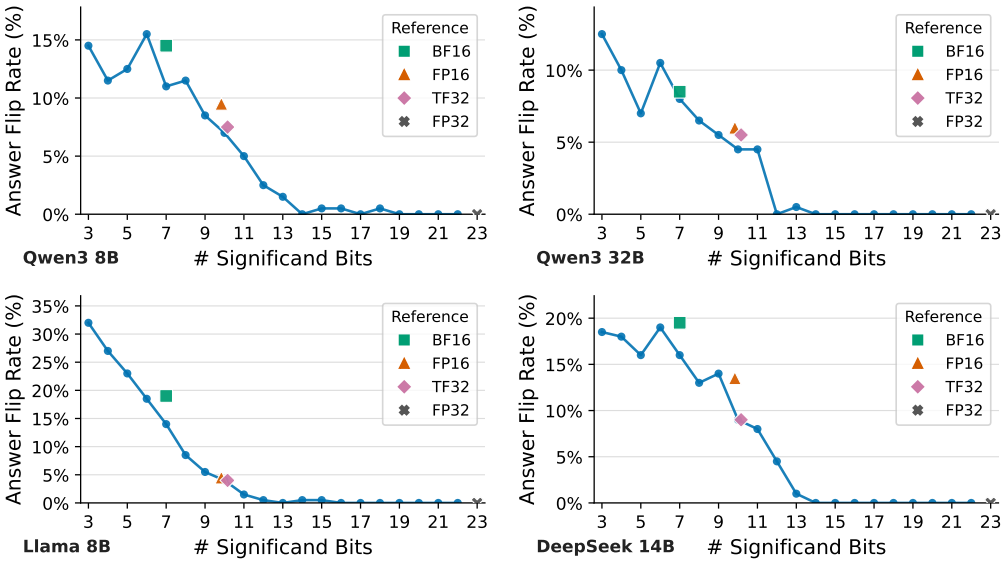


Figure 7: Answer flip rate vs. number of significand bits, complement of Figure 4.

these instances contribute to the overall flip rate, they primarily expose the model's limited capability on these challenging questions rather than systemic precision issues. This observation suggests that accurately measuring numerical reproducibility requires selecting more carefully curated datasets where the model is confident, rather than evaluating on questions that push its capability boundaries.

B.2 Error Contribution Analysis (Figure 2)

Error Decomposition. We first revisit the structural decomposition introduced in §2.2. For decoder layer i, let $\mathbf{H}_i$ denote the pre-Attention residual stream, $\mathbf{A}_i$ denote the post-Attention residual stream, and $\mathbf{H}_{i+1}$ denote the next-layer input after the MLP block. Hatted tensors denote the corresponding

candidate low-precision execution, while unhatted tensors denote the FP32 LayerCast reference. Each layer comprises three primary mappings susceptible to numerical-error injection: the KV-write mapping $\mathbf{H}_i \to (\mathbf{K}_i, \mathbf{V}_i)$, the Attention mapping $(\mathbf{H}_i, \mathbf{C}^{(i)}) \to \mathbf{A}_i$ (where $\mathbf{C}^{(i)}$ is the KV cache consumed by Attention), and the MLP mapping $\mathbf{A}_i \to \mathbf{H}_{i+1}$. We can then decompose the error as:

$$\begin{aligned} \hat{\mathbf{A}}_i - \mathbf{A}_i &= \epsilon^{\mathrm{A}}_{\mathrm{new}} + \epsilon^{\mathrm{A}}_{\mathrm{kv}} + \epsilon^{\mathrm{A}}_{\mathrm{old}}, \\ \hat{\mathbf{H}}_{i+1} - \mathbf{H}_{i+1} &= \epsilon^{\mathrm{M}}_{\mathrm{new}} + \epsilon^{\mathrm{M}}_{\mathrm{old}}, \\ (\hat{\mathbf{K}}_i, \hat{\mathbf{V}}_i) - (\mathbf{K}_i, \mathbf{V}_i) &= \epsilon^{\mathrm{K}}_{\mathrm{new}} + \epsilon^{\mathrm{K}}_{\mathrm{old}}. \end{aligned} \tag{12}$$

Here the superscripts A, M, and K denote Attention, MLP, and KV-write mappings, respectively ϵ_{new} denotes the newly injected local numerical error, ϵ_{old} denotes the error inherited from the mapping input, and ϵ_{kv} denotes the horizontal error inherited through the KV cache.

Error Measurement. We measure these error propagation routes using Teacher Forcing against the H100 FP32 LayerCast reference trajectory, comparing the newly generated candidate tensors against the reference tensors at identical layer and token positions. Using the notation above, for each layer i and token position t, we record the following scalar MSE measurements:

$$E_{\mathrm{in}}(i,t) = \mathrm{MSE}(\widehat{\mathbf{H}}_{i,t}, \mathbf{H}_{i,t}), \tag{13}$$

$$E_{\mathrm{post}}(i,t) = \mathrm{MSE}(\widehat{\mathbf{A}}_{i,t}, \mathbf{A}_{i,t}), \tag{14}$$

$$E_{\mathrm{next}}(i,t) = \mathrm{MSE}(\widehat{\mathbf{H}}_{i+1,t}, \mathbf{H}_{i+1,t}), \tag{15}$$

$$E_{\mathrm{K}}(i,t) = \frac{1}{2}\left[\mathrm{MSE}(\widehat{\mathbf{K}}_{i,t}, \mathbf{K}_{i,t}) + \mathrm{MSE}(\widehat{\mathbf{V}}_{i,t}, \mathbf{V}_{i,t})\right], \tag{16}$$

$$E_{\mathrm{pK}}(i,t) = \frac{1}{t+1}\sum_{s=0}^{t} E_{\mathrm{K}}(i,s). \tag{17}$$

Here E_{in}, E_{post}, and E_{next} measure the accumulated residual-stream error before the Attention, after the Attention, and after the MLP, respectively. E_{K} measures the current-token KV-cache error, while E_{pK} measures the prefix-averaged KV error seen by Attention over $\mathbf{C}^{(i)}_{\le t}$. For the final decoder layer, E_{next} is derived from the output MSE of that layer. We additionally track two local error growth terms:

$$J_{\mathrm{A}}(i,t) = \mathrm{MSE}\left(\widehat{\mathbf{A}}_{i,t} - \widehat{\mathbf{H}}_{i,t}, \mathbf{A}_{i,t} - \mathbf{H}_{i,t}\right), \tag{18}$$

$$J_{\mathrm{M}}(i,t) = \mathrm{MSE}\left(\widehat{\mathbf{H}}_{i+1,t} - \widehat{\mathbf{A}}_{i,t}, \mathbf{H}_{i+1,t} - \mathbf{A}_{i,t}\right). \tag{19}$$

These growth terms, J_{A} and J_{M}, isolate the error increments introduced natively by the Attention and MLP updates. Although these updates operate on tensors already harboring inherited numerical errors, such inherited perturbations are typically orders of magnitude smaller than the original hidden activations. Consequently, the error behavior of the current update is predominantly governed by the original activation values rather than by coherent interactions with errors from preceding layers. This characteristic justifies approximating the accumulated residual-flow MSE by summing the layer-wise local growth MSEs. In practice, utilizing J_{A} and J_{M} provides a more stable estimation of each layer's intrinsic error contribution for our analysis.

Error Propagation Model. For each layer, we fit the replayed token rows with the following first-order approximation:

$$E_{\mathrm{K}}(i,t) \approx \lambda_{\mathrm{K}}(i) E_{\mathrm{in}}(i,t) + \beta_{\mathrm{K}}(i), \tag{20}$$

$$J_{\mathrm{A}}(i,t) \approx \lambda_{\mathrm{in}}(i) E_{\mathrm{in}}(i,t) + \lambda_{\mathrm{pK}}(i) E_{\mathrm{pK}}(i,t) + \beta_{\mathrm{A}}(i), \tag{21}$$

$$J_{\mathrm{M}}(i,t) \approx \lambda_{\mathrm{M}}(i) E_{\mathrm{post}}(i,t) + \beta_{\mathrm{M}}(i). \tag{22}$$

The replay rows contain token-level stochastic variation, so the fitted model estimates the average trend of each propagation relation rather than enforcing a point-wise equality. The slope terms λ describe how inherited residual or prefix-KV error is transported to the next stage, while the intercept terms β estimate layer-local source magnitudes.

Robust Fit. The replay rows occasionally contain extreme MSE outliers, which can make standard least-squares estimators unstable. We therefore estimate the propagation slopes with a Theil–Sen

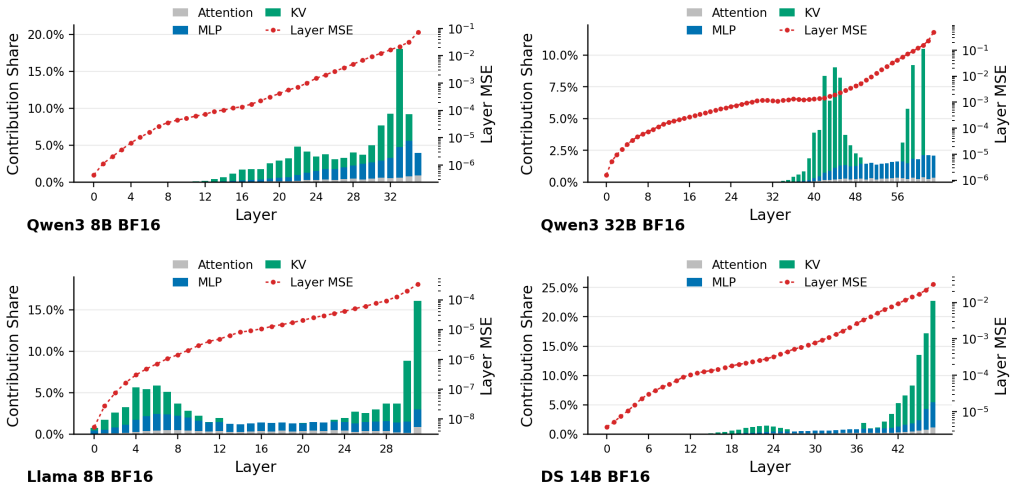


Figure 8: Contribution of error across network depth for the supplementary models, complementing Figure 2.

robust fit [55]. For a single-feature relation, the slope is estimated from the median pairwise trend:

$$\tilde{\lambda} = \text{median}_{r<s,\, \tilde{x}_r \neq \tilde{x}_s} \frac{\tilde{y}_r - \tilde{y}_s}{\tilde{x}_r - \tilde{x}_s}. \tag{23}$$

For the two-feature Attention relation, we estimate the two slopes by alternating Theil–Sen updates until the slopes stabilize. At each iteration, we first subtract the current prefix-KV term and update the residual-stream slope from the remaining residual, then subtract the updated residual-stream term and update the prefix-KV slope from the new residual. This alternating procedure captures the marginal effect of each feature after accounting for the other feature. After the slopes have stabilized, the remaining average offset is used as the fitted intercept β, which represents the local source magnitude not explained by inherited residual or prefix-KV error.

Error Share Attribution. We convert the fitted local source magnitudes into local shares by normalizing them with the expected tensor MSE at the corresponding stage over replay rows:

$$s_{\mathrm{A}}(i) = \frac{\beta_{\mathrm{A}}(i)}{\mathbb{E}_t[E_{\text{post}}(i,t)]}, \qquad s_{\mathrm{K}}(i) = \frac{\lambda_{\mathrm{pK}}(i)\beta_{\mathrm{K}}(i)}{\mathbb{E}_t[E_{\text{post}}(i,t)]}, \qquad s_{\mathrm{M}}(i) = \frac{\beta_{\mathrm{M}}(i)}{\mathbb{E}_t[E_{\text{next}}(i,t)]}. \tag{24}$$

Here s_{A}, s_{K}, and s_{M} denote the local Attention-compute, KV-cache, and MLP-compute shares, respectively. The KV source is multiplied by λ_{pK} because newly written KV error affects the residual stream only after it is consumed by Attention.

We then recursively allocate the final hidden-state error share from the last layer to the first. Let ρ be the currently unexplained share arriving from deeper layers, initialized as $\rho = 1$. At layer i, the MLP contribution is assigned first:

$$c_{\mathrm{M}}(i) = \rho s_{\mathrm{M}}(i), \qquad \rho_{\mathrm{A}}(i) = \rho\,(1 - s_{\mathrm{M}}(i))\,. \tag{25}$$

The remaining share then enters the Attention side, where Attention computes, and KV-cache contributions are assigned as

$$c_{\mathrm{A}}(i) = \rho_{\mathrm{A}}(i) s_{\mathrm{A}}(i), \qquad c_{\mathrm{K}}(i) = \rho_{\mathrm{A}}(i) s_{\mathrm{K}}(i). \tag{26}$$

The unassigned share is propagated to the previous layer:

$$\rho \leftarrow \rho_{\mathrm{A}}(i)\,(1 - s_{\mathrm{A}}(i) - s_{\mathrm{K}}(i))\,. \tag{27}$$

This process assigns the remaining error mass to the local sources encountered along the backward residual flow, and produces the layer-wise attribution scores $c_{\mathrm{A}}(i)$, $c_{\mathrm{M}}(i)$, and $c_{\mathrm{K}}(i)$.

Figure Construction. Figure 2 reports the recursive attribution for Qwen3-14B under native BF16 execution, and Figure 8 reports the corresponding results for the supplementary models. The plotted layer-wise shares are $c_{\mathrm{A}}(i)$, $c_{\mathrm{M}}(i)$, and $c_{\mathrm{K}}(i)$, shown as gray, blue, and green bars for Attention compute, MLP compute, and KV-cache contribution, respectively. The red curve on the secondary axis reports the layer-ending MSE $\mathbb{E}_t[E_{\text{next}}(i,t)]$. We compute these figures on a 32-example MedQA calibration split, with at most 1024 generated tokens replayed per example.

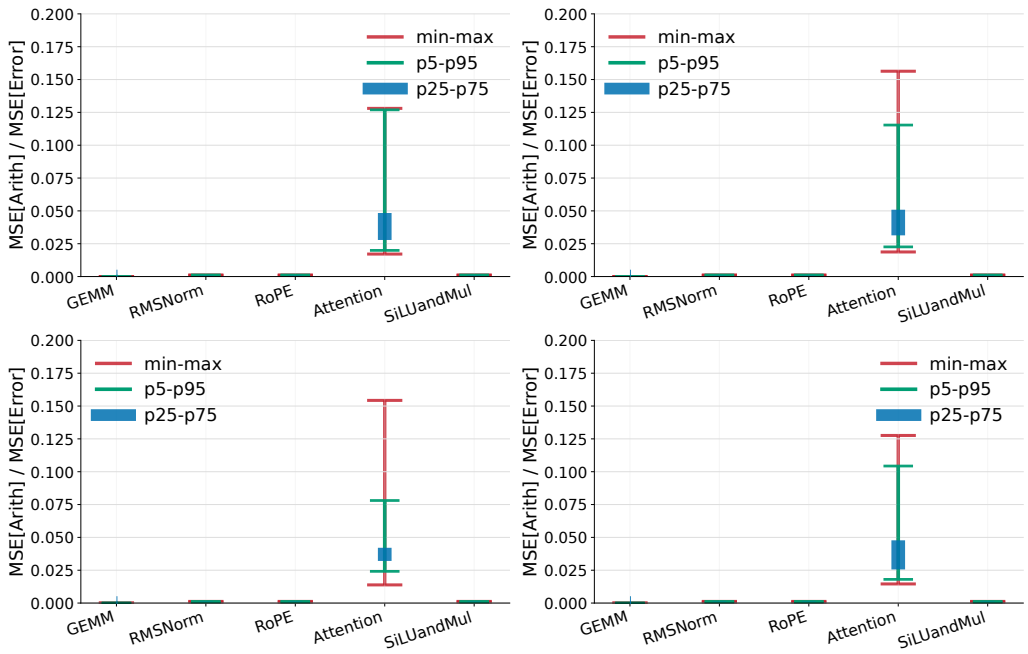


Figure 9: The $\mathbb{E}[\epsilon^2_{\text{arith}}]/\mathbb{E}[(\hat{y}-y)^2]$ ratio for all components, complement of Figure 3.

B.3 Truncation vs Arithmetic (Figure 3)

To quantify the exact source of numerical noise, we empirically isolate the truncation error (ϵ_{trunc}) from the arithmetic error (ϵ_{arith}). Specifically, for any given operator $\mathcal{F}$ and its exact FP32 input x, we evaluate three distinct execution paths:

- **Reference FP32:** $y = \mathcal{F}_{\text{FP32}}(x)$.
- **Native Low-Precision (Total Error):** $\hat{y} = \mathcal{Q}_{\text{BF16}}(\mathcal{F}_{\text{BF16}}(\mathcal{Q}_{\text{BF16}}(x)))$, representing the standard execution utilizing the native vLLM BF16 path.
- **Truncation-Only (Control):** $\tilde{y} = \mathcal{Q}_{\text{BF16}}(\mathcal{F}_{\text{FP32}}(\mathcal{Q}_{\text{BF16}}(x)))$, a strictly controlled implementation where inputs and outputs are quantized, but all internal kernel arithmetic is securely performed in full FP32.

Based on these execution paths, we measure three corresponding Mean Squared Error (MSE) terms: the total error $E = \mathbb{E}[(\hat{y}-y)^2]$, the isolated truncation error $T = \mathbb{E}[(\tilde{y}-y)^2]$, and the direct arithmetic discrepancy $L = \mathbb{E}[(\hat{y}-\tilde{y})^2]$.

Rather than assuming strict additive independence between quantization and arithmetic operations, we employ a two-factor Shapley value decomposition to fairly attribute the final error. In this formulation, we assess the marginal contributions of the arithmetic operations under two different cooperative conditions:

- **Residual Contribution:** The marginal error introduced by the arithmetic operations given that truncation has already occurred, calculated as $(E-T)/E$.
- **Direct Contribution:** The marginal error of the arithmetic operations evaluated directly against the truncation-only baseline, calculated as L/E.

The Shapley arithmetic contribution is the exact average of these two marginal effects. Consequently, the normalized contribution ratio plotted in our analysis (i.e., the y-axis in Figure 3 and 9) is formally derived as:

$$\frac{\text{Shapley[Arith]}}{\text{MSE[Error]}} = \frac{1}{2}\left(\frac{E-T}{E} + \frac{L}{E}\right) = \frac{E-T+L}{2E} \tag{28}$$

To construct the profiling distribution, we systematically evaluate every operator instance across all decoder layers. Specifically, we randomly sample 16 questions from our MedQA 200 subset and

trace the forward passes for the first 64 generated tokens. During the reference FP32 execution, we record the exact input tensors for each operator. We then independently replay these recorded inputs, compute the expected MSE values across all 1,024 token positions (16×64) for all operators, and calculate their corresponding Shapley ratios. The boxplots in Figure 9 illustrate the distribution of these computed ratios across all instances belonging to the same operator family.

As clearly shown in the distribution (Figure 3 and Figure 9 for the additional models), the arithmetic error is negligible for almost all fundamental kernels, with Attention being the sole exception. We break down the root cause for each kernel family below, perfectly corroborating our instruction-level profiling in Table 1:

GEMM. For matrix multiplications, while the input and output tensors are stored in BF16, the internal dot-product accumulation within the Tensor Cores is natively executed in FP32. Consequently, the minor numerical variance introduced by the non-associativity of floating-point accumulation order is completely absorbed by the massive precision loss that occurs when the final 32-bit registers are cast back down to memory storage at the kernel boundary.

RMSNorm, RoPE, and SiLUAndMul. A superficial inspection of the vLLM implementation might suggest that these operations are executed natively in BF16. However, whether it is the reduction and accumulation steps required for variance computation in RMSNorm, or the fundamental arithmetic required for computing activations (SiLUAndMul) and rotational embeddings (RoPE), the actual ALU execution is universally upcast to FP32. Tracing down to the PyTorch [49] ATen bindings (the underlying framework vLLM relies on), we find that applying basic arithmetic to BF16 tensors triggers an automatic upcast to Float32. Specifically, examining the underlying `c10::BFloat16` implementation confirms that the overloaded addition, subtraction, and multiplication operators explicitly cast operands to single-precision:

```
inline C10_HOST_DEVICE BFloat16
operator+(const BFloat16& a, const BFloat16& b) {
  return static_cast<float>(a) + static_cast<float>(b);
}

inline C10_HOST_DEVICE BFloat16
operator-(const BFloat16& a, const BFloat16& b) {
  return static_cast<float>(a) - static_cast<float>(b);
}

inline C10_HOST_DEVICE BFloat16
operator*(const BFloat16& a, const BFloat16& b) {
  return static_cast<float>(a) * static_cast<float>(b);
}
```

This software-level upcasting behavior exactly mirrors our FP32 reference and is consistent across both NVIDIA A100 and H100 architectures. It directly maps to the `FADD` and `FMUL` 32-bit PTX instructions observed in our hardware profiling (Table 1), leaving boundary truncation as the only practical source of numerical divergence.

Attention. The Attention kernel is the distinct exception, exhibiting a non-negligible arithmetic error fraction. This anomaly is driven by a two-fold mechanism. Primarily, unlike standard kernels that strictly defer quantization to the final memory write, the complex multi-stage nature of fused Attention introduces intermediate intra-kernel truncations that do not occur at the kernel boundary. Secondarily, this instability is exacerbated by a compiler-induced behavior where Triton's `tl.dot` operation silently falls back to the TF32 format on modern GPUs. This aggressive fallback restricts the arithmetic precision to only 10 bits and transforms what should be a strict boundary issue into a compounding in-kernel arithmetic error.

In summary, this analysis definitively proves that with the exception of the Attention kernel's intra-kernel truncations and TF32 fallback, end-to-end non-reproducibility is not caused by the internal arithmetic sequence, but is strictly a kernel boundary truncation problem.

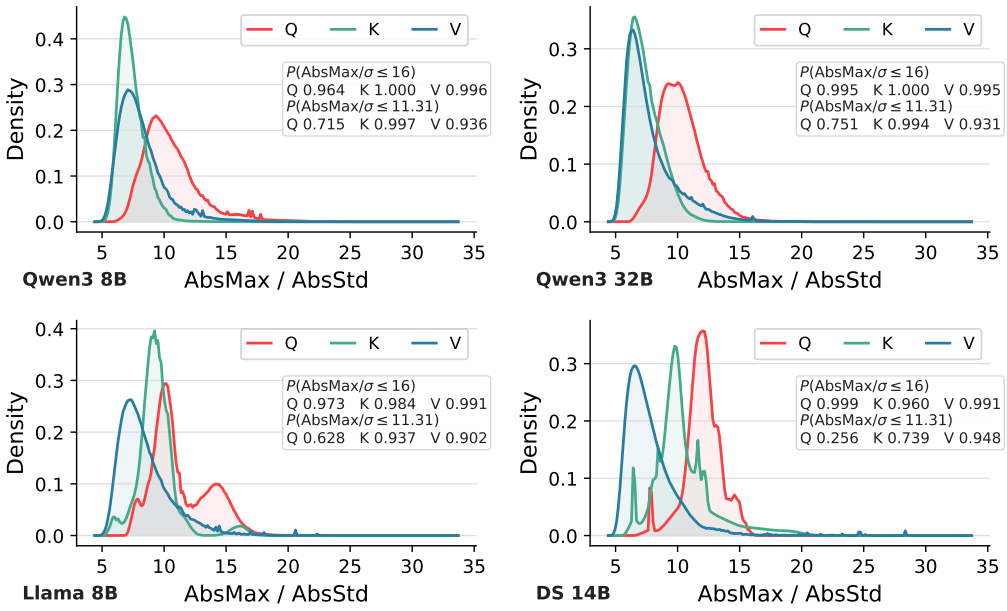


Figure 10: QKV distribution, complement of Figure 5.

B.4 QKV Distribution (Figure 5)

We evaluate the pre-Attention Query (Q), Key (K), and Value (V) tensors under native BF16 execution by computing the ratio of the absolute maximum value to the standard deviation (AbsMax/σ) for each tensor. The corresponding probability density distributions are plotted across all tokens within the complete MedQA 200 subset. Figure 5 and Figure 10 illustrate these distributions.

B.5 Comparison with Quantization Method

Unlike weight-activation and KV-cache quantization, weight-only quantization (*e.g.*, GPTQ [16] and AWQ [39]) compresses only static parameters and can be directly stacked with HEAL. To evaluate this composability, we apply HEAL to W4A16 checkpoints (with group size 128) across all five models on the full MedQA test set [34] (1273 questions) on A100. HEAL runs on the dequantized FP16 weights, which are bit-identical to those multiplied by the native W4A16 kernel, so both paths compute the exact same function. To avoid conflating quantization accuracy loss with reproducibility failure, we anchor each checkpoint against its own weights executed in FP32 with LayerCast [70].

Table 6: Answer flip rate and task accuracy of weight-only quantized models (W4A16) on the MedQA test set (1,273 questions, A100). Flip rates are measured against each checkpoint's own FP32 execution.

Configuration	Qwen3-8B		Qwen3-14B		Qwen3-32B		Llama-8B		DeepSeek-14B	
	Flip Rate	Accuracy	Flip Rate	Accuracy	Flip Rate	Accuracy	Flip Rate	Accuracy	Flip Rate	Accuracy
AWQ (FP32 reference)	—	76.36%	—	80.99%	—	85.00%	—	67.32%	—	72.43%
AWQ (W4A16)	9.43%	75.81%	5.81%	81.78%	6.60%	85.23%	5.66%	68.66%	14.53%	72.43%
AWQ + HEAL	0.31%	76.59%	0.08%	80.99%	0.00%	85.00%	0.16%	67.32%	0.31%	72.43%
GPTQ (FP32 reference)	—	75.57%	—	80.91%	—	86.33%	—	65.91%	—	72.03%
GPTQ (W4A16)	10.68%	75.33%	5.97%	81.38%	7.54%	86.96%	4.40%	66.22%	13.83%	72.35%
GPTQ + HEAL	0.31%	75.49%	0.16%	80.75%	0.39%	86.49%	0.08%	65.91%	0.16%	72.03%

As shown in Table 6, native W4A16 execution still suffers from 4.4–14.5% answer flip rates. Stacking HEAL on the same checkpoints reduces the flip rate to 0–0.39% and lowers the Final Hidden MSE on MedQA-200 from 1.6×10^{-5}–2.1×10^{-4} to 1.2×10^{-8}–9.9×10^{-8}, while keeping task accuracy within 0.25 percentage points of the FP32 reference.

B.6 Model Quality Preservation

To verify that HEAL does not introduce systematic quality degradation, we compare HEAL against the native BF16 baseline on Qwen3-14B across perplexity, general knowledge, instruction following, mathematical reasoning, coding, multilingual understanding, and long-context tasks, alongside reference numbers from the Qwen3 Technical Report [68]. Following the Technical Report (Table 15), downstream benchmarks are evaluated in thinking mode using the official sampling settings, whereas WikiText-2 [43] is evaluated via raw likelihood scoring with thinking disabled. As shown in Table 7, HEAL matches the native BF16 baseline within sampling variance across all benchmarks; the largest deviation (on LiveCodeBench-v5) remains within 1.5 standard errors across its 167 problems and slightly favors HEAL.

Table 7: Model quality of Qwen3-14B under native BF16 and HEAL.

Benchmark / metric	Task	Tech Report	BF16	HEAL
WikiText-2 PPL ↓	Language modeling, full test split	—	11.1106	11.1026
MMLU-Redux Acc. ↑ [18]	General knowledge, 5330 examples	88.6%	89.72%	89.81%
IFEval Strict Acc. ↑ [75]	Instruction following	85.4%	87.06%	86.32%
MATH-500 Pass@1 ↑ [38]	Mathematical reasoning	96.8%	95.6%	95.6%
LiveCodeBench-v5 Pass@1 ↑ [32]	Code generation	63.5%	59.28%	64.67%
MMMLU-14 Acc. ↑ [46]	Multilingual knowledge, fixed 10% subset	77.9%	77.89%	77.84%

To assess long-context behavior, we evaluate a compact RULER [29] subset covering four task categories: needle retrieval (`niah_single_2`), variable tracking (`ruler_vt`), frequent-word extraction (`ruler_fwe`), and multi-hop QA (`ruler_qa_hotpot`). We evaluate two instances per task across six context lengths (48 instances in total) using greedy decoding with thinking disabled. As shown in Table 8, HEAL achieves identical accuracy to native BF16 at every context length.

Table 8: Long-context accuracy (%) of Qwen3-14B on the four-task RULER subset.

	4K	8K	16K	32K	64K	128K	Avg.
BF16	87.50	87.50	83.33	83.33	79.17	79.17	83.33
HEAL	87.50	87.50	83.33	83.33	79.17	79.17	83.33

C HEAL Implementation

HEAL achieves low overhead by improving precision at truncation boundaries while keeping heavy computation on efficient 16-bit Tensor Core paths and avoiding a substantial increase in memory footprint. The implementation is organized as a layered set of precision-protection mechanisms. HEAL-Base first protects the main residual-flow path: hidden states, residual tensors, normalization inputs/outputs, and operator outputs are kept in FP32, while GEMMs still use FP16-path Tensor Core inputs and Attention remains on the native FP16 Attention path. HEAL-Base+Attn builds on this FP32 residual-flow path and adds INT16-based Q/K/V protection for Attention, storing Q/K/V in compact INT16 form and unpacking them into dual-FP16 dot products during Attention computation. HEAL-Base+Attn+Skip is the policy-optimized version of HEAL-Base+Attn: under the error-attribution policy, it skips the extra INT16 Q/K/V protection on a selected number of initial layers, so those layers fall back to the HEAL-Base Attention path rather than native BF16/FP16 execution. Finally, HEAL applies FP16 promotion and algebraic compensation to GEMM inputs on top of HEAL-Base+Attn, stores the KV cache with a grouped INT16 codec, and protects all layers.

HEAL-Base. HEAL-Base keeps the main hidden states, residual stream, normalization inputs/outputs, and operator outputs in FP32, while leaving Attention on the native FP16 path. Its low overhead comes from two factors. First, the dominant GEMM operations still use hardware-optimized Tensor Core kernels: cuBLASLt [11] and hipBLASLt [30] expose GEMMs with 16-bit inputs and FP32 accumulation/output, so HEAL-Base can keep GEMM outputs in FP32 while still relying on optimized vendor-library implementations. Second, the additional FP32 residual-flow operations account for a relatively small fraction of the end-to-end forward pass, so keeping these tensors in FP32 does not dominate serving latency. We also apply a per-weight-matrix power-of-two scaling before GEMM and invert the scale after the GEMM output, which places the weights in a better FP16

representable range and keeps the weight-representation error small. In ShareGPT serving evaluation, HEAL-Base stays within -1.3% to 17.1% TPOT overhead relative to native BF16 across models and hardware platforms. The slightly negative overhead occurs for Qwen3-32B on A100, which we attribute to normal system-level measurement fluctuation and report directly.

HEAL-Base+Attn. HEAL-Base+Attn builds on HEAL-Base and protects the Attention path. We re-implement a Triton version of Attention in which Q/K/V tensors are stored in compact INT16 form with scaling, unpacked into dual-FP16 pieces for the Attention dot products, and written to an FP32 output buffer. The GEMM, normalization, and residual paths remain the same as in HEAL-Base. This design mainly reduces Q/K/V storage truncation error without substantially increasing the memory footprint. Although the dual-FP16 expansion adds extra dot-product work, the overhead is acceptable because Attention is largely memory-bound. In ShareGPT serving evaluation, HEAL-Base+Attn stays within 5.9% to 50.0% TPOT overhead relative to native BF16 across models and hardware platforms.

HEAL-Base+Attn+Skip. HEAL-Base+Attn+Skip is the policy-optimized version of HEAL-Base+Attn. It keeps the same FP32 residual-flow path as HEAL-Base and applies the same INT16-based Q/K/V protection as HEAL-Base+Attn on most layers. Following the error-attribution policy, HEAL-Base+Attn+Skip skips the extra Q/K/V protection on a selected number of initial layers, which fall back to the HEAL-Base Attention path. In ShareGPT serving evaluation, HEAL-Base+Attn+Skip stays within 2.2% to 41.5% TPOT overhead relative to native BF16 across models and hardware platforms.

HEAL. HEAL builds on HEAL-Base+Attn and further applies the FP16 algebraic compensation for GEMM as described in §3. This compensation represents each high-precision activation with two FP16 components and combines two FP16 Tensor Core GEMMs into an FP32 output, thereby recovering the residual information that would be lost by a single FP16 cast. We also implemented an INT8 compensation path, but its quantization overhead offsets the compute benefit of INT8 execution and introduces additional GPU memory cost. Therefore, the final design uses FP16 compensation. In ShareGPT serving evaluation, HEAL ranges from 70.1% to 119.9% TPOT overhead relative to native BF16 across models and hardware platforms.

Grouped INT16 KV Codec. Because a single tensor-wide scale wastes INT16 resolution when a few channels dominate in magnitude, HEAL uses a lightweight grouped encoding for the KV cache. Before quantization, K and V are normalized using fixed per-channel statistics calibrated offline on a small ShareGPT subset (shared across all hardware platforms). Each 128-element K or V vector then stores a shared BF16 base scale alongside compact power-of-two sub-group shifts in a 4-byte metadata header, incurring only a 1.6% storage overhead (520 bytes per KV head per token) over native 16-bit caches.

Error-Attribution Policy. For HEAL-Base+Attn+Skip, we use a small calibration set of 8 MedQA examples (replaying up to 1024 generated tokens per example) to determine which layers can safely skip INT16 Q/K/V protection. Using the median Final Hidden MSE as our validation metric to reduce sensitivity to extreme-error outliers, we require the accepted policy to stay within a $1.2\times$ median-MSE budget relative to HEAL-Base+Attn. For each model, we evaluate two candidate skipping orders: a *contribution order* that skips the k lowest-contribution layers based on our error-attribution analysis, and a *prefix order* that skips the initial k layers. We select the largest k that satisfies the budget and pick the order with the lower median MSE. In our calibration, prefix policies are consistently selected (reported in Table 9), indicating that while error attribution reliably identifies which mechanism to optimize, coarse layer-level skipping works best on the earliest layers.

Table 9: Number of skipped layers in HEAL-Base+Attn+Skip.

Configuration	**Qwen3-8B**	**Qwen3-14B**	**Qwen3-32B**	**Llama-8B**	**DeepSeek-14B**
HEAL-Base+Attn+Skip	11	14	13	5	15

Fine-Grained Channel Policy. When extending this skipping policy to the complete HEAL pipeline, we find that because HEAL drives the overall residual error close to the FP32 noise floor, layer-level error attribution becomes too noisy to reliably guide whole-layer GEMM skipping without degrading precision. Instead of dropping entire layers, HEAL supports a finer-grained per-layer policy that restricts dual-FP16 GEMM compensation in selected MLP projections to the top input channels

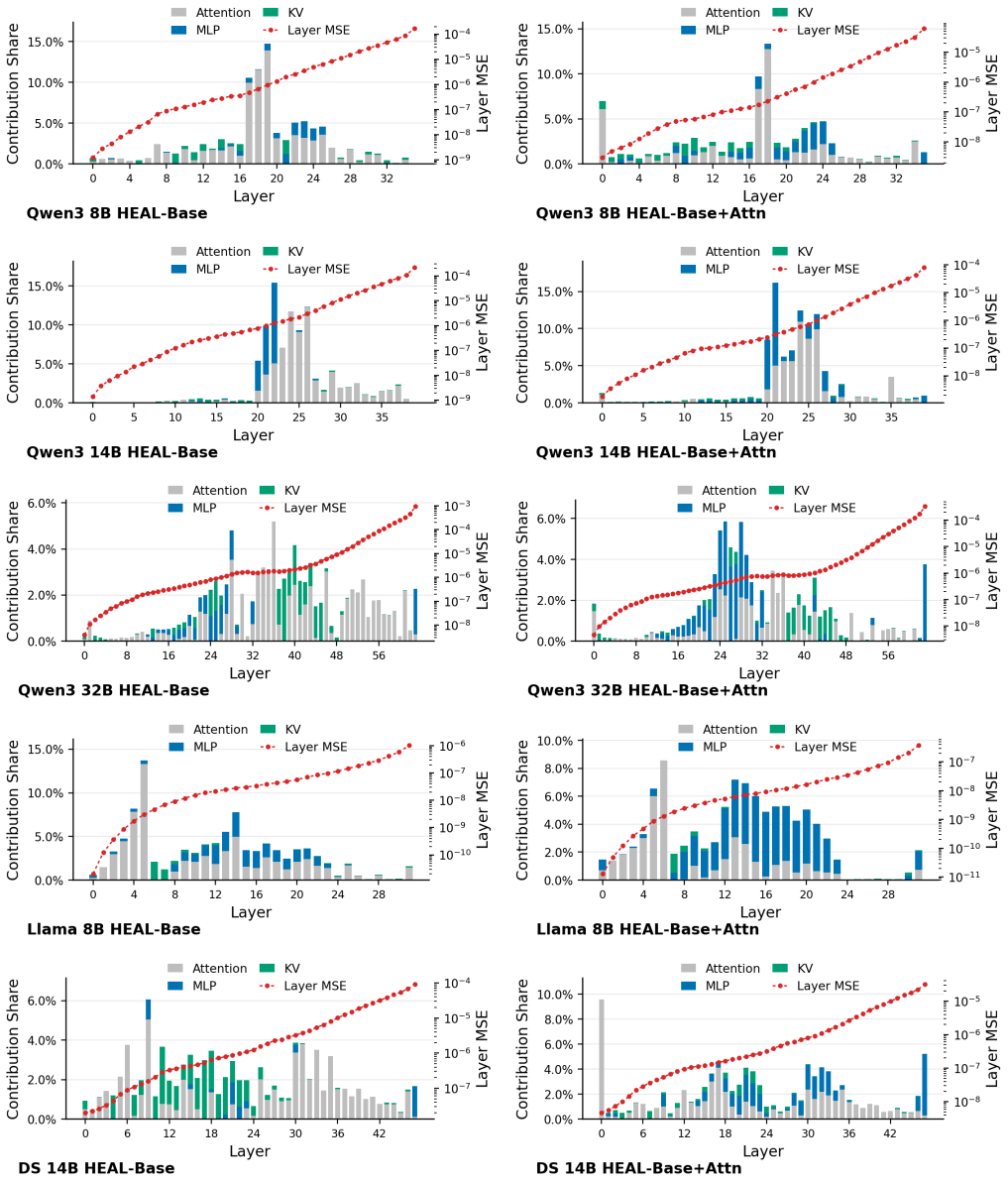


Figure 11: Error-contribution maps for HEAL-Base and HEAL-Base+Attn across all evaluated models.

ranked by residual energy: on Qwen3-14B, compensating an average of 42% of input channels across 39 of the 80 MLP projections saves 12.7% of GEMM FLOPs and 7.6% of H100 TPOT, while adding only $\sim$0.2 percentage points to the answer flip rate on MCR-Bench. For maximum precision, our default HEAL configuration applies full compensation across all layers.

Contribution Diagnostics under HEAL Variants. We also report the error-contribution maps for HEAL-Base and HEAL-Base+Attn in Figure 11. These plots illustrate the staged optimization path of HEAL: after HEAL-Base broadly reduces residual-boundary truncation, the remaining error becomes concentrated in attention and KV-cache propagation, motivating the Q/K/V protection in HEAL-Base+Attn; after HEAL-Base+Attn protects the attention path, the relative MLP/GEMM contribution becomes more visible, motivating the GEMM compensation in HEAL.

D Additional Performance Results

Per-Model Breakdown. In addition to the serving setup described in Appendix A, each GPU–model pair is evaluated over three independent runs. Tables 10, 11, 12, and 13 report the mean TPOT alongside the standard error of the mean for the remaining models.

Table 10: ShareGPT TPOT and Memory Usage for the **Qwen3-8B** model.

Configuration	A100 TPOT (ms)	H100 TPOT (ms)	MI300X TPOT (ms)	Memory Usage (GiB)
BF16	62.4 ± 0.6	25.2 ± 0.1	33.6 ± 0.1	15.8
TF32 (+LayerCast)	147.9 ± 0.5	67.3 ± 0.0	74.9 ± 0.2	16.6
FP32 (+LayerCast)	747.5 ± 4.5	304.2 ± 1.4	139.7 ± 0.2	16.6
HEAL-Base	70.9 ± 0.2	29.5 ± 0.1	36.4 ± 0.0	16.0
HEAL-Base+Attn	78.4 ± 0.2	34.3 ± 0.1	50.4 ± 0.1	16.0
HEAL-Base+Attn+Skip	76.2 ± 0.1	32.9 ± 0.0	46.3 ± 0.1	16.0
HEAL	136.1 ± 0.1	54.0 ± 0.4	69.4 ± 0.1	16.2

Table 11: ShareGPT TPOT and Memory Usage for the **Qwen3-32B** model.

Configuration	A100 TPOT (ms)	H100 TPOT (ms)	MI300X TPOT (ms)	Memory Usage (GiB)
BF16	119.3 ± 0.6	49.1 ± 0.2	118.0 ± 0.2	61.9
TF32 (+LayerCast)	268.1 ± 3.5	149.2 ± 0.4	289.7 ± 0.8	64.0
FP32 (+LayerCast)	617.0 ± 3.5	338.2 ± 0.5	555.4 ± 1.0	64.0
HEAL-Base	117.7 ± 0.4	51.7 ± 0.2	126.0 ± 0.2	62.3
HEAL-Base+Attn	126.3 ± 0.8	56.9 ± 0.2	162.2 ± 0.7	62.4
HEAL-Base+Attn+Skip	121.9 ± 1.5	55.7 ± 0.4	154.8 ± 0.2	62.3
HEAL	205.8 ± 1.4	83.5 ± 1.7	255.8 ± 0.5	62.7

Table 12: ShareGPT TPOT and Memory Usage for the **Llama-8B** model.

Configuration	A100 TPOT (ms)	H100 TPOT (ms)	MI300X TPOT (ms)	Memory Usage (GiB)
BF16	63.8 ± 0.3	25.0 ± 0.0	32.8 ± 0.1	15.5
TF32 (+LayerCast)	149.1 ± 0.5	66.4 ± 0.0	74.0 ± 0.1	16.4
FP32 (+LayerCast)	752.9 ± 1.4	311.2 ± 0.5	137.6 ± 0.3	16.4
HEAL-Base	69.7 ± 0.4	28.6 ± 0.1	35.5 ± 0.1	15.6
HEAL-Base+Attn	77.2 ± 0.3	32.7 ± 0.1	48.4 ± 0.2	15.6
HEAL-Base+Attn+Skip	75.5 ± 0.3	32.3 ± 0.4	46.4 ± 0.1	15.6
HEAL	136.3 ± 0.7	52.6 ± 0.2	68.6 ± 0.2	15.8

Cost–Reproducibility Trade-off. Table 14 summarizes the overall cost–reproducibility trade-off across all five models and three GPUs. The answer flip rate is averaged across the five models (from Table 2), while TPOT and base memory footprint are normalized to the native BF16 baseline on the same model and GPU (summarizing Table 3 and Tables 10–13). All HEAL configurations remain close to the native BF16 memory footprint: model weights and runtime workspace grow by only 0.6–1.3% for the ablation variants and 1.3–2.5% for HEAL (compared with 3.4–5.8% for the 32-bit LayerCast baselines), while the grouped INT16 KV codec adds only 1.6% storage overhead over native 16-bit caches (Appendix C).

Deployment Guidance. For mission-critical deployments requiring strict cross-hardware reproducibility, we recommend the complete HEAL pipeline, which achieves FP32-level flip rates (0.07%) at 1.70–2.20$\times$ the BF16 TPOT (versus 4.16–13.11$\times$ for full FP32). For latency-constrained deployments that can tolerate a $\sim$1% residual flip rate, HEAL-Base+Attn and HEAL-Base+Attn+Skip reduce the BF16 flip rate by $\sim$10$\times$ at $\leq$1.50$\times$ the BF16 TPOT, while HEAL-Base alone reduces flips by $> 6\times$ at $\leq$1.17$\times$ TPOT. The fine-grained channel policy in Appendix C further provides intermediate operating points between these lightweight variants and full HEAL.

Table 13: ShareGPT TPOT and Memory Usage for the **DS-14B** model.

Configuration	A100 TPOT (ms)	H100 TPOT (ms)	MI300X TPOT (ms)	Memory Usage (GiB)
BF16	115.1 ± 0.7	44.6 ± 0.1	59.6 ± 0.1	28.2
TF32 (+LayerCast)	270.0 ± 1.1	120.4 ± 0.0	137.1 ± 0.4	29.4
FP32 (+LayerCast)	1405.1 ± 4.2	580.6 ± 1.6	258.5 ± 0.2	29.4
HEAL-Base	124.3 ± 0.2	50.2 ± 0.2	64.1 ± 0.2	28.4
HEAL-Base+Attn	137.1 ± 0.7	57.2 ± 0.3	86.2 ± 0.2	28.4
HEAL-Base+Attn+Skip	132.9 ± 0.7	54.9 ± 0.0	79.4 ± 0.2	28.4
HEAL	246.2 ± 0.4	96.9 ± 0.4	124.8 ± 0.4	28.7

Table 14: Cost–reproducibility trade-off across five models and three GPUs. Brackets indicate the range across models (for flip rate and memory) or across all 15 model–GPU pairs (for TPOT).

Configuration	Flip Rate	TPOT vs. BF16	Memory vs. BF16	Role
BF16	11.47% [8.03%, 16.53%]	1.00×	—	Native baseline
TF32 (+LayerCast)	2.64% [1.97%, 3.77%]	2.47× [2.23×, 3.04×]	+3.4–5.8%	32-bit baseline
FP32 (+LayerCast)	0.03% [0.00%, 0.11%]	8.85× [4.16×, 13.11×]	+3.4–5.8%	High-precision reference
HEAL-Base	1.77% [1.22%, 2.39%]	1.10× [0.99×, 1.17×]	+0.6–1.3%	Ablation
HEAL-Base+Attn	1.02% [0.66%, 1.25%]	1.30× [1.06×, 1.50×]	+0.6–1.3%	Ablation
HEAL-Base+Attn+Skip	1.15% [0.84%, 1.48%]	1.25× [1.02×, 1.42×]	+0.6–1.3%	Ablation
HEAL	**0.07% [0.06%, 0.10%]**	**2.08× [1.70×, 2.20×]**	**+1.3–2.5%**	**Recommended**

E Detailed Description of MCR-Bench

Within MCR-Bench, we curate 4500 test cases drawn from 10 established datasets: 1500 each from the medical, legal, and financial domains, covering more than 20 use cases. We select tasks that are consequential, where an inconsistent answer carries real cost, and objectively evaluable, with a discrete, parseable answer so that a change in output is a well-defined event. Questions are sampled with a fixed seed from the evaluation pools of the constituent datasets, without model-specific filtering. To evaluate answer flip rates, each configuration is executed once on each GPU with greedy decoding and up to 4096 generated tokens. An answer flip is counted whenever the parsed final answer differs from that of the reference FP32 execution, with parse failures and length truncations treated as a distinct answer. Specifically, the Final Hidden MSE is measured on a 100-question subset of MCR-Bench, where the token-averaged error already stabilizes.

Medical Domain. We construct the 1500-question medical subset from MedQA-USMLE-4-options [34], CareQA [4], and PubMedQA [35] to cover complementary forms of clinically meaningful reasoning. We used questions that require multi-step reasoning over symptoms, laboratory findings, imaging, treatment options, or study conclusions, and excluded items primarily based on isolated factual recall. The resulting subset spans 9 mission-critical use cases: **emergency diagnosis, critical care management, sepsis and severe infection identification, cardiovascular emergencies, pediatric emergency recognition, toxicology, adverse drug reactions, procedural decision-making, and medical ethics in emergencies**. These settings are particularly important for evaluating robustness because model errors may lead not only to incorrect answers, but also to unsafe recommendations that threaten patient safety.

Legal Domain. The 1500-question legal subset is drawn from CaseHOLD [73], LegalBench (insurance policy interpretation) [23], BarExamQA [74], and LexGLUE (UNFAIR-ToS) [7], covering 6 practical legal reasoning use cases: **contract risk review, civil rights analysis, criminal procedure, information protection, statutory interpretation, and insurance coverage analysis**. These use cases are mission-critical because small reasoning perturbations can materially alter conclusions regarding rights, obligations, liability, or access to remedies. They also stress reasoning patterns that are especially sensitive to numerical instability, including close semantic distinctions, rule application under ambiguity, and conflicts between superficially plausible choices.

Finance Domain. The 1500-question finance subset is constructed from Financial Sentiment Analysis [41, 42, 42], FinQA [8], and FinClaim [57], organized into 5 use cases: **financial claim credibility, market sentiment monitoring, numerical reasoning over tables, risk and exposure identification, and classification of financially meaningful language**. These tasks target high-stakes settings where model outputs inform investment analysis, risk monitoring, compliance review, or

financial decision support, and where subtle instability can cause models to miscalculate figures, misread disclosures, confuse speculation with evidence, or misclassify claim credibility under uncertainty.

F Existing Assets Used

In accordance with the NeurIPS guidelines for responsible research, we disclose all existing models, software libraries, and datasets utilized in this work. We acknowledge the original creators and adhere to their respective licenses and terms of use.

F.1 Models

- **Qwen3 (8B, 14B, 32B)** [68]: Developed by Alibaba Cloud. The model weights and architecture are utilized for our numerical divergence analysis. We additionally use the W4A16 checkpoints `Qwen/Qwen3-{8B,14B,32B}-AWQ` (AWQ [39], released by Qwen) and `RedHatAI/Qwen3-{8B,14B,32B}-quantized.w4a16` (GPTQ [16]). *License: Apache 2.0*. URL: `https://huggingface.co/collections/Qwen/qwen3`
- **Llama-3.1-8B-Instruct** [22]: Developed by Meta. Utilized for zero-shot and reasoning evaluations. We additionally use the W4A16 checkpoints `hugging-quants/Meta-Llama-3.1-8B-Instruct-AWQ-INT4` (AWQ) and `RedHatAI/Meta-Llama-3.1-8B-Instruct-quantized.w4a16` (GPTQ). *License: Llama 3.1 Community License*. URL: `https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct`
- **DeepSeek-R1-Distill-Qwen-14B** [24]: Developed by DeepSeek. Utilized to evaluate the impact of distillation and reinforcement learning on numerical stability. We additionally use the W4A16 checkpoints `casperhansen/deepseek-r1-distill-qwen-14b-awq` (AWQ) and `RedHatAI/DeepSeek-R1-Distill-Qwen-14B-quantized.w4a16` (GPTQ). *License: MIT License*. URL: `https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-14B`

F.2 Software and Frameworks

- **vLLM (v0.19.0)** [37]: An open-source high-throughput inference engine serving as the foundation for our experimental testbed. *License: Apache 2.0*. URL: `https://github.com/vllm-project/vllm`
- **PyTorch** [49]: Deep learning framework underlying vLLM. *License: BSD 3-Clause*. URL: `https://github.com/pytorch/pytorch`
- **FlashAttention-3** [56]: Used as the highly optimized exact Attention backend. *License: BSD 3-Clause*. URL: `https://github.com/Dao-AILab/flash-attention`
- **Triton** [61]: An open-source programming language for writing highly efficient custom GPU kernels. *License: MIT License*. URL: `https://github.com/triton-lang/triton`
- **CUDA Toolkit (12.8)** & **cuBLASLt** [11]: Proprietary libraries developed by NVIDIA for GPU acceleration. *License: NVIDIA Software License Agreement*. URL: `https://developer.nvidia.com/cuda-toolkit`
- **ROCm (7.2.1)** & **hipBLASLt** [30]: Open-source software stack and BLAS library for AMD GPUs. *License: MIT License*. URL: `https://github.com/ROCm/rocm`
- **LM Evaluation Harness (v0.4.9)** [17]: Used for WikiText-2, MMLU-Redux, IFEval, and RULER in the model quality evaluation. *License: MIT License*. URL: `https://github.com/EleutherAI/lm-evaluation-harness`
- **LightEval (v0.10.0)** [25]: Used for MATH-500 and LiveCodeBench in the model quality evaluation. *License: MIT License*. URL: `https://github.com/huggingface/lighteval`

F.3 Datasets

We utilized the following publicly available datasets for system benchmarking and to construct the Mission-Critical Reproducibility Benchmark (MCR-Bench).

System Profiling Datasets:

- **ShareGPT V3** [1]: Used for evaluating Time Per Output Token (TPOT) and peak memory overhead. *License: Apache 2.0.* URL: https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered
- **C4 Dataset** [52]: A cleaned version of Common Crawl used to profile the magnitude of structural outliers in intermediate activations. *License: Apache 2.0.* URL: https://www.tensorflow.org/datasets/catalog/c4

MCR-Bench Datasets (Medical Domain):

- **MedQA** [34]: USMLE multiple-choice questions. *License: MIT License (original release); CC BY 4.0 (Hugging Face release).* URL: https://huggingface.co/datasets/GBaker/MedQA-USMLE-4-options
- **CareQA** [4]: Healthcare evaluation benchmark. *License: Apache 2.0.* URL: https://huggingface.co/datasets/HPAI-BSC/CareQA
- **PubMedQA** [35]: Biomedical research question answering. *License: MIT License*. URL: https://github.com/pubmedqa/pubmedqa

MCR-Bench Datasets (Legal Domain):

- **CaseHOLD** [73]: Multiple-choice legal holdings. *License: Apache 2.0.* URL: https://github.com/reglab/casehold
- **LegalBench** [23]: Legal reasoning benchmark. *License: CC BY 4.0.* URL: https://huggingface.co/datasets/nguha/legalbench
- **BarExamQA** [74]: Reasoning-focused legal retrieval benchmark. *License: CC BY-SA 4.0.* URL: https://huggingface.co/datasets/reglab/barexam_qa
- **LexGLUE** [7]: Legal language understanding benchmark; we use the UNFAIR-ToS and CaseHOLD tasks. *License: CC BY 4.0.* URL: https://huggingface.co/datasets/coastalcph/lex_glue

MCR-Bench Datasets (Finance Domain):

- **Financial Sentiment Analysis** [42, 41]: Financial text classification combining Financial PhraseBank and the FiQA sentiment task. *License: CC0 (combined release); CC BY-NC-SA 3.0 (Financial PhraseBank); not specified (FiQA).* URL: https://www.kaggle.com/datasets/sbhatti/financial-sentiment-analysis
- **FinQA** [8]: Numerical reasoning over financial data. *License: MIT License*. URL: https://github.com/czyssrs/FinQA
- **FinClaim** [57]: Evidentiary reasoning of financial claims. *License: CC BY-SA 4.0.* URL: https://huggingface.co/datasets/ImBinarry/FinClaimRadar-Financial-Reasoning

Model Quality Benchmarks:

- **WikiText-2** [43]: Language modeling perplexity. *License: CC BY-SA*. URL: https://huggingface.co/datasets/EleutherAI/wikitext_document_level
- **MMLU-Redux 2.0** [18]: General knowledge. *License: CC BY 4.0.* URL: https://huggingface.co/datasets/edinburgh-dawg/mmlu-redux-2.0
- **IFEval** [75]: Instruction following. *License: Apache 2.0.* URL: https://huggingface.co/datasets/google/IFEval
- **MATH-500** [38]: Mathematical reasoning. *License: MIT License (PRM800K and MATH).* URL: https://huggingface.co/datasets/HuggingFaceH4/MATH-500
- **LiveCodeBench** [32]: Code generation (v5). *License: MIT License*. URL: https://huggingface.co/datasets/livecodebench/code_generation_lite

- **MMMLU** [46] and **MMLU** [27]: Multilingual and English knowledge. *License: MIT License*. URL: `https://huggingface.co/datasets/openai/MMMLU`, `https://huggingface.co/datasets/cais/mmlu`
- **RULER** [29]: Long-context tasks, instantiated with HotpotQA [69] (*License: CC BY-SA 4.0*) and Paul Graham essays as filler text (*license not specified*). URL: `https://github.com/NVIDIA/RULER`, `https://huggingface.co/datasets/hotpotqa/hotpot_qa`

G Limitations

While HEAL substantially improves cross-hardware reproducibility, we note the following limitations:

- **Hardware Dependence:** The performance efficiency of our approach relies on flexible, hardware-optimized GEMM implementations.
- **Functional, not Bit-wise:** HEAL bounds numerical errors to substantially reduce the probability of output divergence, but it does not provide strict bit-to-bit determinism.
- **Model-Dependent Calibration:** The layer-skipping policy of HEAL-Base+Attn+Skip and the per-channel KV-cache normalization statistics of HEAL are calibrated per model, making the optimization model-dependent.
- **Performance Trade-offs:** While significantly more efficient than global FP32 upcasting, HEAL still incurs a 70–120% TPOT overhead (about 2.1$\times$ the BF16 TPOT on average) relative to the native BF16 baseline.
- **FP8 Formats:** Our current implementation targets 16-bit formats, as native FP8 execution discards too much precision at kernel boundaries to achieve HEAL-level reproducibility alone. Extending HEAL to FP8 would introduce higher overhead.

H Broader Impacts

This paper addresses the critical challenge of non-reproducibility in LLM inference, where heterogeneous hardware and dynamic software configurations often lead to divergent outputs even under supposedly deterministic settings like greedy decoding. Our proposed solution ensures consistent model behavior across diverse infrastructure and multiple runs. This work significantly enhances the reliability of LLMs for mission-critical applications such as medical diagnosis, legal reasoning, and production workflow automation, where reproducibility is a fundamental requirement. Consequently, this research fosters greater trust in LLM systems deployed in complex, real-world datacenter environments.

I Acknowledgment of LLM Usage

We used OpenAI Codex, Claude Code, and Google Gemini as assistive tools during the preparation of this work. OpenAI Codex and Claude Code were used to assist with writing experiment scripts and running evaluations. Codex was also used to assist with implementing the system solution, while Claude Code was additionally used to draft parts of this manuscript. Google Gemini was used for copyediting and refining to improve clarity and readability. All LLM-generated content was reviewed and verified by the authors, who take full responsibility for the final manuscript.