nnTrace: Detecting and Localizing Silent Bugs in Distributed Training

Haitian Jiang[1]* **Shaowei Zhu**[2] **Zhen Zhang**[2] **Zhenyu Song**[2]
Xinwei Fu[2] **Zhen Jia**[2] **Yida Wang**[2] **Jinyang Li**[1]
[1]New York University [2]Amazon
{haitian.jiang, jinyang}@nyu.edu

Abstract

Distributed training is essential for scaling LLM training across thousands of GPUs. However, as distributed training requires complex implementations, they are prone to silent bugs, which do not produce explicit error signals but lead to incorrect training outcomes. Common debugging practices based on monitoring training loss or gradient norm curves are slow, often unable to detect bugs, and provide little help in localizing their source.

We design and implement nnTrace, the first systematic differential testing system for detecting and localizing silent bugs in distributed training. nnTrace aligns intermediate tensors from distributed training with those from a trusted reference implementation. To compare aligned floating-point tensors, nnTrace dynamically estimates per-tensor numerical tolerances using machine-epsilon-scale perturbations of the reference run, guided by local-sensitivity analysis. This allows nnTrace to distinguish bug-induced deviations from benign numerical differences. Experimental results demonstrate that nnTrace effectively detects 11 existing bugs and 3 new bugs in the widely used Megatron-LM framework. nnTrace is effective in various training recipes, including low-precision recipes involving BF16 and FP8. Notably, Megatron-LM has already adopted the method proposed by nnTrace in its development workflow. Our code is available at https://github.com/haitian-jiang/nnTrace.

1 Introduction

The pursuit of scale has driven the training of ever larger models with billions to trillions of parameters [27, 9], which have demonstrated state-of-the-art capabilities in many domains [64, 59]. Consequently, modern training pipelines have become sophisticated large-scale distributed systems, with a myriad of distributed training strategies, such as data parallelism [24], tensor parallelism [47], pipeline parallelism [36], and distributed optimizer [44]. Supporting a customizable combination of all these techniques has led to significant implementation complexity in distributed training systems.

The complexity in distributed training causes reliability issues in the form of a growing number of silent bugs [65, 49] in these systems. Silent bugs [3, 39, 40, 38] refer to bugs that do not produce crashes, hangs, or other explicit error signals, but lead to an incorrect training outcome that is unexpected by the system developer. Such silent bugs may stem from wrong data [3, 40], or incorrect communication operations for cross-device synchronization [39, 38] in the training framework. These bugs may degrade the quality of the trained model, thereby wasting significant training time and massive GPU resources. Therefore, it is critical to detect and localize silent bugs in distributed training programs before the actual training commences.

*Work partially completed during an internship at Amazon.

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

Given a distributed training program, how do we tell whether it is correct? A common approach is differential testing: developers compare the candidate distributed implementation with a trusted reference on a smaller model, looking for agreement between their loss and gradient norm curves. These aggregate comparisons can be slow to reveal a bug. As illustrated in Figure 1 with bug 1 in Table 1, it can take over 4,000 iterations (more than 6 hours) for two curves to show a 3% discrepancy. Once a bug is suspected, localizing its source requires comparing intermediate tensors against the reference, because errors can propagate across many layers. Distributed implementations reorder and partition tensors, making it difficult to align a reference tensor with its constituent shards in the candidate. Even after alignment, deciding whether the tensors are "close enough" requires tolerances that are hard to set, especially with low-precision data types like BF16 [22] and FP8 [34, 27, 41].

In this work, we propose nnTrace, a streamlined differential testing system for detecting and localizing silent bugs in distributed training. nnTrace turns today's slow, coarse-grained debugging loop into a fast and fine-grained testing workflow. Given a candidate distributed implementation and a trusted reference, nnTrace runs both for a single training iteration rather than thousands. It automatically collects intermediate tensors and aligns them across distributed layouts with minimal user annotation, enabling bug detection and localization at PyTorch module granularity. To compare aligned floating-point tensors, nnTrace dynamically estimates per-tensor numerical tolerances from the reference implementation's response to machine-epsilon-scale perturbations. This allows nnTrace to distinguish bug-induced deviations from benign numerical differences while testing directly at native BF16 and FP8 precision.

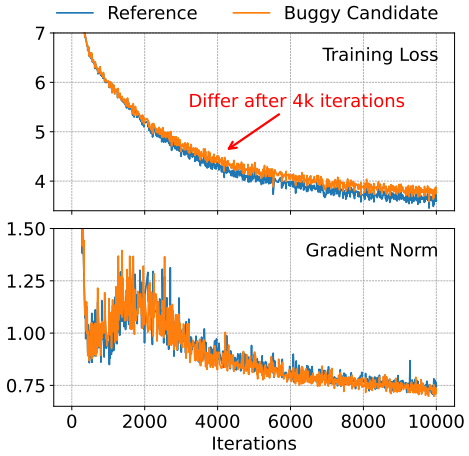

Figure 1: The loss and gradient norm curve of a correct and buggy training. The bug can only be observed after 4000 iterations from training loss curve.

nnTrace is designed to integrate with practical distributed training frameworks such as Megatron-LM. Framework users can apply nnTrace to debug their own distributed training configurations, while framework maintainers can incorporate its methodology into testing pipelines to guard against regressions that affect correctness or numerical stability. Notably, Megatron-LM maintainers have already adopted nnTrace's methodology in their development and testing workflow.

In summary, this work makes the following contributions:

- We design and implement nnTrace, the first differential testing system for silent bug detection and localization in diverse practical training settings, including those with FP8 precision, making silent bug detection fast and actionable.
- We propose a dynamic per-tensor tolerance estimation method, guided by local numerical sensitivity analysis, that distinguishes benign floating-point differences from bug-induced deviations.
- We conduct a comprehensive evaluation of nnTrace on Megatron-LM, detecting 14 silent bugs, including 3 previously unknown bugs that have been confirmed by NVIDIA developers.

2 Problem Formulation and Challenges

Differential testing setup. nnTrace targets the same testing regime commonly used in practice: developers compare a distributed candidate implementation against a trusted reference on a test configuration before launching full-scale training. The test model is often a down-scaled version of the production model so that it can be run by the reference on a single device, while also preserving the relevant distributed features when run by the distributed candidate implementation. This is a coverage assumption: nnTrace can detect bugs only when the test configuration exercises the faulty code path. Bugs triggered only by production-scale tensor shapes or size-dependent kernel selection may require larger test configurations. In this case, nnTrace may use a pipelined reference to accommodate models that do not fit on a single device, as discussed in Appendix B.5.

Given this setup, systematic differential testing poses three technical challenges.

Execution-path coverage and diagnosability. Differential testing must exercise buggy execution paths that can fail in production while still providing enough information to localize the failure. This creates a tension. End-to-end training runs cover integration behavior such as sharding transition, gradient synchronization, loss scaling, and optimizer updates, but a bug introduced early in the computation can propagate through many downstream tensors, obscuring its source. Conversely, testing each module in isolation gives cleaner comparisons, but can bypass cross-module interactions where many silent bugs arise. nnTrace addresses this challenge by first comparing fine-grained traces from the actual training iteration, then using module-wise input rewriting to suppress propagated errors and identify the first faulty component.

Tensor Alignment. Distributed training partitions a model and intermediate tensors across devices using a combination of parallelism strategies, such as pipeline parallelism (PP), data parallelism (DP), tensor parallelism (TP), sequence parallelism (SP), and context parallelism (CP). Consequently, tensors in the candidate exist as shards reordered and fragmented across devices and need to re-assembled according to the parallelization strategy in use in order to align them with tensors in the reference implementation (Figure 3). The alignment depends on the specific parallelism configuration which is not readily available in manually parallelized frameworks such as Megatron-LM. §3 describes how nnTrace constructs this mapping.

Numerical Error Tolerance. Floating-point operations are rounded and non-associative. As a result, two mathematically equivalent implementations can produce numerically different tensors even when both are correct. These benign numerical differences may then accumulate through downstream computation, especially in modern training recipes that use low-precision formats such as BF16 and FP8. Some systems bypass this challenge by running the candidate in higher precision, such as FP32, or by converting computation to finite fields [55]. However, this can reduce bug coverage because many silent bugs are triggered only on native low-precision code paths, such as bugs 7, 8, and 13 in Table 1. Therefore, nnTrace needs a principled way to estimate expected numerical differences at native precision; we present such a method in §4.

3 nnTrace System Design

3.1 System Components

Overview. Figure 2 illustrates the workflow of nnTrace. **Step 1**, the user provides lightweight annotations describing how parameters and activations are partitioned across parallel dimensions (see Appendix B for concrete examples). nnTrace uses these annotations to construct a *canonical mapping* that establishes the correspondence between tensor shards in the candidate and reference implementation for further comparison. **Step 2**, nnTrace estimates the expected numerical error tolerance for the given model using the reference implementation as described in Algorithm 1. **Step 3**, nnTrace runs both the reference and candidate implementations for one training iteration, employing a *consistent distributed tensor generator* to supply identical inputs and a *trace collector* to capture activations, gradients, parameters during forward and backward propagation and parameter updates. **Step 4**, the *equivalence checker* compares the two traces against the estimated tolerance, flagging tensors whose deviations exceed the threshold. **Step 5**, when a bug is detected, nnTrace further localizes the root cause via *module-wise testing*. It rewrites the inputs to every module with consistent distributed tensors. This decoupling prevents error from propagating and eliminates accumulated numerical deviations, enabling precise bug localization.

Canonical Mapping. To correspond and compare tensors between the candidate and reference, nnTrace introduces *logical full tensor*: a tensor assembled from physical shards across devices that is expected to match a physical non-sharded tensor in the reference implementation (Figure 3). nnTrace derives the canonical mapping between a tensor's physical shard and its position in the logical tensor from the user-provided annotations. During trace collection, it reconstructs each logical full tensor from its shards. For parallelism strategies that distribute tensors to devices without sharding them (DP, PP), nnTrace assigns each tensor a *canonical identifier* to align them across data batches and pipeline stages. For models too large to fit on a single device, nnTrace runs the reference with pipeline parallelism with the same mapping mechanism(Appendix B.5).

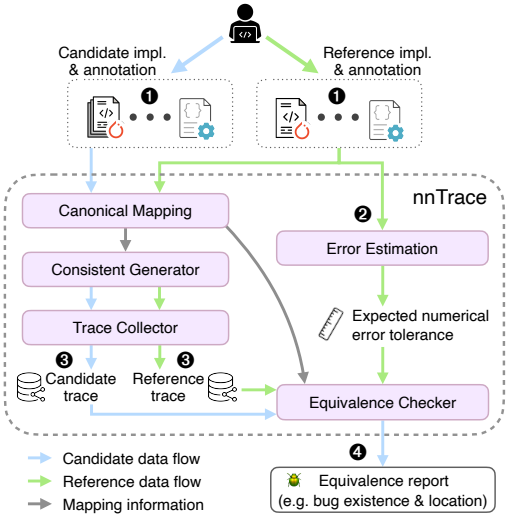

Figure 2: The system overview of nnTrace.

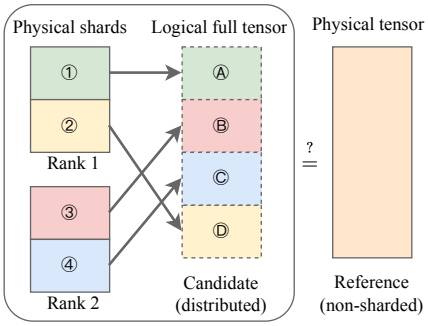

Figure 3: Canonical shard mapping.

Trace Collector, Consistent Tensor Generator, and Rewrite. nnTrace records activations, gradients, and parameters during training to produce a trace for comparison. To ensure that the candidate and reference start from identical logical tensors, nnTrace uses a consistent tensor generator: it creates the same logical full tensor for both runs and then shards the candidate tensor according to the canonical mapping. For localization, nnTrace also supports module-wise input rewrite. After an end-to-end discrepancy is detected, nnTrace reruns individual modules while replacing their inputs with matched logical tensors in both the candidate and reference, thereby preventing discrepancy from propagating. This rewrite is also important for MoE models. Small numerical differences can cause the candidate and reference to dispatch tokens to different experts. nnTrace therefore rewrites the router's input with the reference's value, so that all runs make the same routing decisions, while the experts and all other modules compute on each run's own inputs. Routing outputs themselves are traced and tested as usual, so router-side bugs (e.g., bug 6 in Table 1) are detected through the regular workflow.

Equivalence Checker. After both traces are collected, the equivalence checker determines whether they agree. It first aggregates the distributed tensor shards to reconstruct the corresponding logical full tensors; during this reconstruction, it also checks that tensors replicated across devices hold identical values, which directly catches bugs such as a missing all-reduce that leaves replicas divergent. It then performs differential testing as described in Algorithm 1.

The system implementation details of the components are provided in Appendix B.

3.2 Error Tolerance Estimation and Differential Testing

Comparing aligned tensors requires distinguishing benign floating-point differences from deviations caused by bugs. Their numerical scale depends on the operations and reduction orders used to compute each tensor, so tracking every contribution analytically is impractical. Fixed absolute or relative tolerances chosen from experience can also miss this variation, as we show in §5.3.

Algorithm 1 Error Tolerance Estimation and Differential Testing. The TraceFwd and TraceBwd functions record intermediate tensors.

Require: Reference $\hat{F}$, candidate $\hat{H}$, input X, safety margin $k = 3.5$

// Error tolerance estimation

$\{Y_l\} \leftarrow \text{TraceFwd}(\hat{F}, X)$ ▷ get activation tensors

$G_L \leftarrow \frac{\partial \text{loss}}{\partial Y_L}$ ▷ Y_L: logits

$\{G_l\} \leftarrow \text{TraceBwd}(\hat{F}, G_L)$ ▷ get gradient tensors

$\{T_i\} \leftarrow \{Y_l\} \cup \{G_l\}$ ▷ indexed by layer & type

Sample ΔX with $\|\Delta X\| = 0.5\,\varepsilon_{\text{mch}} \cdot \|X\|$

$\{Y'_l\} \leftarrow \text{TraceFwd}(\hat{F}, X + \Delta X)$; $G'_L \leftarrow \frac{\partial \text{loss}}{\partial Y'_L}$

Sample ΔG with $\|\Delta G\| = 0.5\,\varepsilon_{\text{mch}} \cdot \|G'_L\|$

$\{G'_l\} \leftarrow \text{TraceBwd}(\hat{F}, G'_L + \Delta G)$

$\{T'_i\} \leftarrow \{Y'_l\} \cup \{G'_l\}$

for each tensor index i **do** ▷ $0/0 := 0$, $x/0 := \infty$

  $\tau_i \leftarrow \|T'_i - T_i\| \,/\, \|T_i\|$

// Differential testing

$\{Y''_l\} \leftarrow \text{TraceFwd}(\hat{H}, X)$; $G''_L \leftarrow \frac{\partial \text{loss}}{\partial Y''_L}$

$\{G''_l\} \leftarrow \text{TraceBwd}(\hat{H}, G''_L)$

$\{T''_i\} \leftarrow \{Y''_l\} \cup \{G''_l\}$

for each tensor index i **do**

  $e_i \leftarrow \|T''_i - T_i\| \,/\, \|T_i\|$

  if $e_i > k \cdot \tau_i$ **then** flag T''_i as a bug

To estimate this scale, nnTrace uses the reference run's response to small perturbations as a **perturbation-based proxy**. Algorithm 1 describes the procedure, with $\hat{F}$ and $\hat{H}$ denoting the reference and candidate implementations. The estimation phase perturbs the reference input, namely the embedding output, and the gradient of the logits that initializes backpropagation, then measures the relative norm of each activation or gradient difference. The testing phase compares the corresponding candidate and reference tensors, flagging differences that exceed the estimated tolerance by the safety margin. The algorithm shows forward and backward propagation for clarity; in practice, nnTrace traces complete training iterations, including optimizer updates, and can therefore detect optimizer bugs such as bugs 5 and 9 in Table 1.

We determine the perturbation magnitude and the estimated tolerance in relative terms. For the perturbation, our goal is to match the scale of floating-point roundoff. For values in the normal range, floating-point rounding satisfies $|\hat{x} - x|/|x| \leq u = 0.5\,\varepsilon_{\text{mch}}$ [11], where $\hat{x}$ is the floating-point representation of real number x. Therefore, we choose ΔX so that $\|\Delta X\| = 0.5\,\varepsilon_{\text{mch}}\|X\|$, using this machine-epsilon-scale perturbation as a proxy for roundoff; the seed perturbation ΔG has the same relative norm. We also estimate tolerances in a relative form to handle the varying tensor norms across different layers and types (e.g., activations and gradients). This allows us to directly compare the differences across modules, as well as against the machine epsilon. A tensor whose estimated tolerance is zero is flagged if the candidate's tensor differs from the reference's at all, and a tensor whose non-finite entries (NaN or infinity) differ from the reference's is flagged regardless of the norms.

When conducting the differential testing on the candidate, nnTrace uses a safety margin k to absorb the variability between the estimated and true numerical errors, as the estimate reflects the expected magnitude of numerical errors rather than a strict upper bound. The argument in §4.2 motivates comparing the observed difference with a fixed multiple of the perturbation response. We use $k = 3.5$, calibrated empirically between the largest observed-to-estimated ratio of correct runs and the smallest ratio of the detected injected faults (Appendix C.2).

4 Error Tolerance Analysis

Correct training implementations can produce different tensor values because their arithmetic uses different precisions and reduction orders. The size of these differences depends on both the rounding introduced locally and its propagation through the model. We first compare computed tensors with an ideal real-arithmetic model to explain the depth dependence of these errors, then use the reference implementation's perturbation response to estimate their scale for each tensor.

4.1 Mean-Square Error Analysis

We use a randomly initialized L-layer transformer with pre-layer normalization (Pre-LN) as a tractable model, with zero-mean projection weights, normalization gains near one, and zero biases. Each residual branch then has a bounded expected squared norm and a zero expected inner product with the incoming state. The expected squared norm of the residual stream therefore grows linearly with depth, up to fixed factors. We assume that, on average over tokens, the variances used by LayerNorm are large compared with its stabilizer, and that near the inputs being compared no token's variance falls far below the expected per-coordinate second moment. The resulting local Lipschitz bound for LayerNorm decreases with depth beyond the first layer, limiting error amplification in deeper branches.

Our error model assumes numerically stable local computations: each introduces rounding at machine-epsilon scale relative to its typical output. Local errors may be correlated, but should not persistently reinforce the accumulated error. We also assume that, beyond the first few layers, average amplification is mild enough that early rounding errors do not dominate the total. These conditions describe typical numerical behavior; they do not require independent tokens.

Using Frobenius norms, we normalize the root mean square (RMS) error by the RMS norm of the ideal tensor:

$$\mathcal{E}(\hat{T}, T) := \sqrt{\frac{\mathbb{E}\|\hat{T} - T\|^2}{\mathbb{E}\|T\|^2}}.$$

Here T is the ideal tensor and $\hat{T}$ its computed counterpart; we call $\mathcal{E}$ the RMS relative error. Expectations are over initialization, inputs, and the rounding model, with averaging performed before normalization. Constants below may depend on the architecture, precisions, and reduction scheme, but not on layer index or depth within this regime.

Theorem 1 (Activation Relative Error)**.** *Let Y_l and $\hat{Y}_l$ be the ideal and computed residual states. Under the preceding stability and moderate-amplification conditions,*

$$\mathcal{E}(\hat{Y}_l, Y_l) = O\left(\sqrt{l}\,\varepsilon_{mch}\right).$$

The local mean-square error bound grows with the residual stream's second moment. Summing these contributions, normalizing by the activation's second moment, and taking a square root gives the stated upper bound. For local errors of comparable relative scale with small expected cross terms, this accumulation mechanism also predicts the rising trend observed in several configurations of Figure 4.

For backpropagation on fixed saved forward values, let $\bar{\delta}_l$ be the exact result and $\hat{\delta}_l$ its floating-point evaluation. The calculation starts from the output gradient, called the backward seed. We assume backpropagation amplifies accumulated error by at most a fixed factor more than it amplifies the gradient itself over any part of the backward path.

Theorem 2 (Rounding Error in Backpropagation)**.** *Under the local rounding model and this propagation comparison,*

$$\mathcal{E}(\hat{\delta}_l, \bar{\delta}_l) = O\left(\sqrt{L - l + 1}\,\varepsilon_{mch}\right).$$

Gradients closer to the input collect more rounding from more backpropagation operations. A complete comparison also includes changes to the saved activations, the derivatives used by backpropagation, and the loss gradient caused by forward errors. The proxy therefore perturbs the forward computation to measure this dependence and the backward seed to probe how perturbations propagate through the backward pass.

A parameter gradient is a sum of activation–gradient outer products, so it inherits errors from both factors. We assume factor perturbations remain controlled and that, compared with the gradient itself, their products with the factors do not add up much more coherently across tokens; limited cancellation in the outer-product sum and moderate fluctuations of the squared norms suffice for this. We further assume the backpropagation formulas respond stably to perturbations of the saved forward values, preserving their relative error scale.

Theorem 3 (Parameter Gradient Relative Error)**.** *For ideal and computed parameter gradients G^{W^l} and $\hat{G}^{W^l}$ of a weight matrix, bias, or normalization gain of layer l, these conditions and numerically stable gradient computation give*

$$\mathcal{E}(\hat{G}^{W^l}, G^{W^l}) = O\left(\sqrt{L}\,\varepsilon_{mch}\right)$$

uniformly in l, in the small-error regime.

The forward contribution accumulates from the input and the backward contribution from the output. Together they give a common upper envelope across layers, allowing individual curves to vary beneath it. These bounds explain the broad error scale; their constants still depend on the tensor and its computation, making a per-tensor estimate necessary.

4.2 Perturbation Response as an Error Estimate

The ideal real-arithmetic tensor is unavailable during testing. Instead, for every traced tensor, nnTrace measures

$$e_T = \|T^H - T^F\|, \qquad \tau_T = \|T^{F,+} - T^F\|,$$

where T^H comes from the candidate, T^F from the reference, and $T^{F,+}$ from the perturbed reference. The latter uses machine-epsilon-scale input perturbations and, for backward tensors, a backward-seed perturbation after recomputing the loss gradient on the perturbed forward values. Both quantities use the same relative normalization in the implementation, which cancels in e_T/τ_T.

A small input change can alter rounding decisions throughout the reference run. Its response therefore contains both the propagated input change and local arithmetic differences, just as the candidate comparison contains differences introduced by its kernels and reductions. A first-order Taylor expansion describes both comparisons through the same Jacobians, provided the remainder is small and discrete choices such as expert routing are held fixed.

The proxy is informative when its local differences are sufficiently large in mean square along the error directions present in a correct candidate. Suppose that contributions from distinct error sources have zero expected cross terms, conditional on the ideal computation, and that the proxy's local differences dominate the candidate's in mean square along every direction up to a fixed constant. Then, to first order, the mean-square candidate difference is at most that constant times the mean-square proxy response. The Jacobians need not be computed explicitly: the reference run measures their effect. The comparison constant can still change when implementations have different rounding patterns, such as additional storage casts in tensor parallelism, or FP8 scaling factors that the candidate computes from tensor shards instead of the full tensor (Appendix E.7); the delayed scaling with amax reduction used in our experiments gives all shards the factor of the full tensor.

Since an actual check uses one realization, nnTrace includes a safety margin and flags $e_T > k\tau_T$ for inspection. We use $k = 3.5$, which lies between the largest ratio observed in correct runs and the smallest ratio of the detected injected faults (Appendix C.2). This margin is calibrated empirically; the mean-square analysis does not bound every realized ratio. Appendix E gives the precise assumptions and derivations underlying this argument.

5 Evaluation

To evaluate the practicality and effectiveness of nnTrace, we focus on the following research questions: whether it detects real silent bugs in production frameworks and how efficient it is (§5.1); whether its perturbation-based error estimation accurately tracks the true numerical errors across diverse models and configurations (§5.2); what is the advantage of dynamic error estimation compared to common threshold-based practices (§5.3). We also show the efficiency of nnTrace by comparing its bug detection time against the approach mentioned in §1, and the tracing time for large models where the reference implementation cannot run on a single card, with details in Appendix C.4.

We selected Megatron-LM [37] as our primary target because it is one of the most widely-used production-ready training frameworks. It has been adopted and adapted by major technology companies for large model training, such as NVIDIA [37], Microsoft [35], Amazon [2], ByteDance [21]. We tested popular language model architectures, including GPT, Llama, Mamba, and MoE models with BF16 and FP8 precisions.

5.1 Detected Silent Bugs

Table 1 shows the silent bugs that nnTrace discovered. We collected the silent bugs from the issue page of Megatron-LM and TransformerEngine. Although most of the silent bugs are typically resolved by engineers during the development stage, we found 11 silent bugs and reproduced them using the corresponding history commits. nnTrace was able to discover all of them, with the safety margin $k = 2$ (Appendix C.1). The types and impact of these bugs are described in Table 1. These bugs will cause the training to have unintended behaviors and cause consequences like degraded model accuracy.

We also conducted a sweep test on different combinations of 4D parallelism (DP, PP, TP, CP) on Megatron-LM and TransformerEngine. nnTrace successfully detected 3 new silent bugs during the test. They have all been confirmed by NVIDIA developers, and 2 of them have already been fixed. The 14 bugs span three categories: wrong data, wrong communication, and missing communication, covering the major silent bugs in distributed training. More details on the categorization is in Appendix A.1.

New Bug Case Study In this section, we highlight nnTrace's ability to uncover subtle silent bugs in training frameworks through a case study on a new bug discovered. Bug 13 corrupts the gradients of self-attention activations when context parallelism is enabled. The root cause is that a non-contiguous input tensor was passed into a customized attention kernel, which silently operated on an invalid memory layout. As the bug resides inside an opaque customized kernel rather than framework-level Python code, it is invisible to approaches that operate on computation graphs.

Table 1: List of silent bugs discovered by nnTrace. The first 11 are existing bugs collected from the issue list of Megatron-LM and TransformerEngine. The last 3 are found by nnTrace.

ID	Type	Description	Impact	Issue	Fix
1	WD	TP: wrong embedding mask	Wrong forward, gradients	N/A	pull/80
2	WD	AR: wrong input	Wrong gradients	371	`fc2c81d`
3	WD	CP: wrong loss scaling	Wrong gradients	673	`0c2cd58`
4	WD	DP: wrong loss scaling	Wrong gradients	906	`3bdcbbb`
5	WC	ZeRO: embedding and LM-head untied	Wrong parameter update	656	`db2040f`
6	MC	SP: router weights not synchronized	Wrong gradients	599	pull/619
7	WC	TP: wrong FP8 communication group	Wrong forward, gradients	335	pull/442
8	WD	AR: wrong tensor by FP8 cast	Wrong loss	539	pull/646
9	WC	ZeRO: parameter update failure	No parameter update	663	`75120db`
10	WD	PP: wrong stage division	Wrong model get trained	468	`8e98a72`
11	WC	TP: wrong gradients with overlap	Wrong gradients	1616	pull/1554
12	MC	SP: layernorm weights not synchronized	Wrong gradients	1446	pull/1528
13	WD	CP: wrong attention gradients	Wrong gradients	1557	pull/1539
14	WC	TP+CP: wrong layernorm gradients	Wrong gradients	1677	N/A

WD: Wrong Data **WC**: Wrong Communication **MC**: Missing Communication **AR**: Activation Recomputation
Issue: The numbers are issue IDs for the bugs; N/A means the bug is not reported in an issue.
Fix: The hex string is the fixing commit id; the numbers are the fixing pull requests; N/A means the bug has not been fixed.

nnTrace pinpointed it by localizing the discrepancy to the output of the attention gradient module. The case studies on other new bugs are detailed in Appendix A.3.

Efficiency To show that nnTrace can detect silent bugs within reasonable time, we conducted the time cost comparison experiment for nnTrace against the common debugging practice mentioned in §1. We use bug 1 [3] as an example. The common approach requires training both the reference and candidate implementation until the loss values diverge significantly. As shown in Figure 1, it took more than 4000 iterations over a duration of 6 hours and 40 minutes for the loss curve to exhibit a 3% relative error. In contrast, nnTrace detects the bug from a single training iteration in 54 seconds.

5.2 Accuracy of Numerical Error Estimation

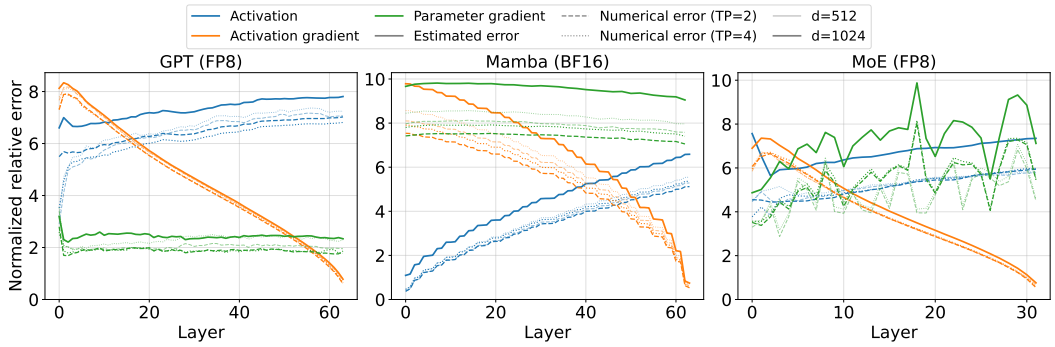

Figure 4: Estimated numerical error (solid) versus observed numerical error from tensor parallel runs. Three representative configurations selected from a sweep over architectures and precisions. The y-axis is the relative error normalized by ε_{mch} of BF16, including FP8 runs, as the mixed-precision configuration stores the tensors in BF16 (Appendix E.7).

To assess the numerical error estimate, we compare the perturbation response with observed differences from correctly implemented distributed candidates. The sweep covers architectures (GPT, Llama, Mamba, MoE), precisions (BF16, FP8), tensor parallelism of sizes 2 and 4, context parallelism of size 2, hidden dimensions 512 and 1024, and 32 or 64 layers (16 or 32 for MoE); Appendix C.1 gives the configuration.

Figure 4 illustrates three representative combinations of model architectures and numerical precisions, with comprehensive results detailed in Appendix C.3. For clarity, each subplot contrasts the estimated numerical error tolerance for a specific hidden dimension against observed numerical errors across various parallelism and hidden dimension configurations. Across the evaluated settings,

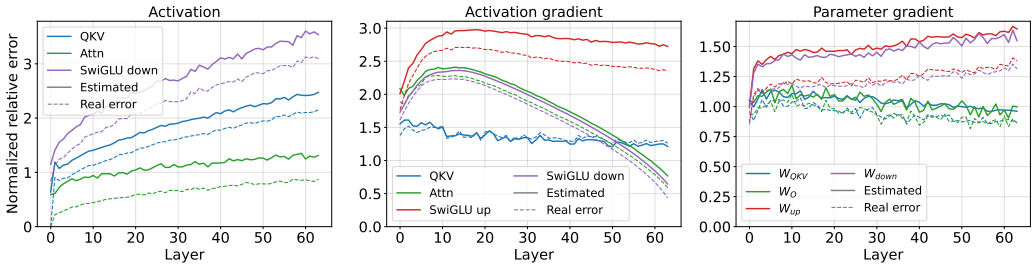

Figure 5: Per-module estimation accuracy for a 64-layer Llama (BF16, $d = 1024$) configuration. The observed numerical error is from tensor parallelism of size 2. The y-axis is normalized by ε_{mch} of BF16. Colors account for different modules and line styles account for estimated or real numerical error.

Table 2: nnTrace's dynamic tolerances versus fixed `torch.allclose` tolerances. "False detected" is the fraction of compared tensors flagged in the correct tensor-parallel candidates; false negatives count the injected faults, out of 96, for which no tensor is flagged.

`atol`	`rtol`	False detected	False negative
0	10^{-5}	98.3%	0/96
10^{-8}	10^{-5}	98.3%	0/96
10^{-5}	10^{-2}	46.7%	0/96
10^{-4}	3×10^{-2}	40.2%	0/96
10^{-3}	5×10^{-2}	39.7%	5/96
10^{-2}	10^{-1}	37.0%	27/96
5×10^{-2}	2×10^{-1}	19.5%	37/96
nnTrace estimation ($k = 3.5$)		0.0%	4/96

nnTrace's estimation tracks the observed numerical error. The estimated and observed errors vary with layer depth, with different accumulation patterns for forward and backward tensors. The rising activation and falling activation-gradient trends are qualitatively consistent with the model in §4.1; the full activation-gradient errors also include the effect of perturbing the saved forward values. Parameter-gradient curves can vary considerably between layers, particularly for MoE, whose expert gradients depend on the tokens routed to each expert, so the theory's uniform upper bound should not be read as a prediction of a flat curve. Within the tested configurations, the plotted per-layer errors vary little between tensor-parallel sizes 2 and 4. The few tensors with the largest ratios, all under tensor parallelism of size 4, are discussed in Appendix C.3.

Figure 5 resolves the observed and estimated errors by module for a Llama configuration. The variation between modules motivates per-tensor tolerances: modules differ in both the rounding introduced by their operations and the amplification of incoming perturbations. The estimated curves track this variation, with corresponding GPT results in Appendix C.3.

5.3 Comparison with Static Error Thresholds

As indicated by Figures 4 and 5, numerical errors vary significantly across layer counts, data types, and model architectures, so a single static tolerance is ill-suited to intermediate tensor comparisons. We compare nnTrace's dynamic estimation against `torch.allclose` with a sweep of fixed `rtol`/`atol` values on a dataset of 96 simulated bugs across four architectures (GPT, LLaMA, Mamba, MoE), each of which adds a random perturbation of relative norm 10, 50, or 100 times BF16 ε_{mch} to the output of one or all decoder layers in the forward pass, or to their output gradients in backpropagation. These faults have the form of the proxy's perturbations, only larger, and are injected in single-device runs; nnTrace uses Algorithm 1 with $k = 3.5$.

Table 2 shows the trade-off of the tested fixed thresholds on this suite: strict tolerances flag benign numerical errors, while loose tolerances miss injected faults; non-zero `atol` also masks errors in small gradient tensors (observed in bug 13). Fixed thresholds on the relative error that nnTrace uses, chosen from the other configurations, still miss 5 to 37 of the faults.

5.4 False Positives and False Negatives

nnTrace raised no false alarms on the correct candidates of Table 2 or of the sweep in §5.2: the largest observed-to-estimated ratio was 2.87, below the safety margin (Appendix C.2). This is an empirical property of the tested configurations, consistent with the agreement between estimated and observed numerical errors in §5.2 and with the conditional analysis in §4.

In terms of false negatives, nnTrace may miss bugs whose deviations are at the magnitude of per-iteration numerical error. In Table 2, nnTrace misses four faults, all of one kind: the smallest forward fault, which perturbs the output of the first layer by $10\,\varepsilon_{\text{mch}}$ in relative norm, in each FP8 configuration. Correct FP8 candidates already differ from the reference by about 5 to 13 ε_{mch} in the layer outputs, and FP8 quantization enlarges small differences sublinearly (Appendix E.7), so this fault changes the downstream tensors by at most 2.4 times the estimated tolerance, within the range of correct-run ratios. The same faults are detected in BF16, where correct candidates differ by 1 to 2 ε_{mch}. In addition, nnTrace can also miss nondeterministic bugs such as race conditions that cannot be reproduced within one iteration. Hardware failures such as silent data corruption are out of scope.

6 Related Work

ML system correctness and verification. Extensive research targets ML system correctness through testing frameworks [61, 20] and compilers [46, 32] using API fuzzing [42, 13, 52, 54, 57, 62, 63] or graph generation [30, 29, 25, 50, 33], often with differential testing as a building block. Verification-based approaches [19, 1, 28, 65, 53] prove semantic consistency of graph rewrites or parallelization but struggle with bugs below their abstractions and require costly, carefully maintained rule sets. Two recent works target silent errors in distributed training from different angles: TrainCheck [20] synthesizes invariants from correct traces and, lacking a reference, misses bugs that produce consistent but wrong values (e.g., bugs 13, 14 in Table 1); D3 [51] fuzzes distributed frameworks and checks an equivalence rule on final predictions, but as a black-box approach cannot localize bugs or detect errors that do not manifest at the output. nnTrace complements both by conducting single-iteration fine-grained tensor comparisons with a dynamic tolerance, enabling precise detection and localization of intermediate bugs.

Floating-point error analysis and Lipschitz properties. Probabilistic analyses of FP error accumulation [6, 18, 15] and rigorous FP semantics modeling [12, 48, 7] provide theoretical foundations but are expensive to scale to practical models. nnTrace instead uses the reference's response to small perturbations as a proxy for round-off error accumulation. Local Lipschitz bounds [45, 43] describe how layers amplify errors; while dot-product attention is not globally Lipschitz [23], local bounds hold under mild assumptions [4, 23, 14]. Using perturbation responses as tolerances additionally requires the directional moment comparison of §4.2, and the safety margin is calibrated empirically.

7 Conclusion

nnTrace is the first differential testing system for detecting and localizing silent bugs in distributed training. nnTrace compares fine-grained traces from an actual candidate training iteration against a trusted reference, while reconstructing and aligning tensors that are sharded across distributed parallelism strategies. To distinguish benign floating-point differences from bug-induced deviations, nnTrace dynamically estimates per-tensor tolerances from machine-epsilon-scale perturbations of the reference run. When a discrepancy is found, nnTrace uses module-wise input rewriting to suppress error propagation and localize the faulty component. Our evaluation on Megatron-LM and TransformerEngine shows that nnTrace detects 14 silent bugs, including 3 previously unknown bugs confirmed by NVIDIA developers, across practical BF16 and FP8 training settings.

Acknowlegement

Haitian Jiang and Jinyang Li are partially funded by grants from NVIDIA and AMD.

References

[1] Jai Arora, Sirui Lu, Devansh Jain, Tianfan Xu, Farzin Houshmand, Phitchaya Mangpo Phothilimthana, Mohsen Lesani, Praveen Narayanan, Karthik Srinivasa Murthy, Rastislav Bodik, Amit Sabne, and Charith Mendis. TensorRight: Automated Verification of Tensor Graph Rewrites. *Artifact for "TensorRight: Automated Verification of Tensor Graph Rewrites"*, 9(POPL):29:832–29:863, January 2025.

[2] AWS. Amazon sagemaker model training. https://aws.amazon.com/sagemaker-ai/train/, 2023.

[3] AWS. [bugfix] shard embedding hooks. https://github.com/awslabs/slapo/pull/80, 2023.

[4] Valérie Castin, Pierre Ablin, and Gabriel Peyré. How Smooth Is Attention? In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pages 5817–5840. PMLR, 2024.

[5] Michael P. Connolly, Nicholas J. Higham, and Theo Mary. Stochastic rounding and its probabilistic backward error analysis. *SIAM Journal on Scientific Computing*, 43(1):A566–A585, 2021.

[6] Fredrik Dahlqvist, Rocco Salvia, and George A Constantinides. A probabilistic approach to floating-point arithmetic. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 596–602. IEEE, 2019.

[7] Arnab Das, Ian Briggs, Ganesh Gopalakrishnan, Sriram Krishnamoorthy, and Pavel Panchekha. Scalable yet rigorous floating-point error analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20, pages 1–14, Atlanta, Georgia, November 2020. IEEE Press.

[8] Pablo de Oliveira Castro, El-Mehdi El Arar, Eric Petit, and Devan Sohier. Error analysis of sum-product algorithms under stochastic rounding, 2025. arXiv:2411.13601v2, revised August 2025.

[9] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[10] El-Mehdi El Arar, Devan Sohier, Pablo de Oliveira Castro, and Eric Petit. Bounds on non-linear errors for variance computation with stochastic rounding. *SIAM Journal on Scientific Computing*, 46(5):B579–B599, 2024.

[11] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM computing surveys (CSUR)*, 23(1):5–48, 1991.

[12] Eric Goubault. Static Analyses of the Precision of Floating-Point Operations. In Patrick Cousot, editor, *Static Analysis*, pages 234–259, Berlin, Heidelberg, 2001. Springer.

[13] Qianyu Guo, Xiaofei Xie, Yi Li, Xiaoyu Zhang, Yang Liu, Xiaohong Li, and Chao Shen. Audee: Automated testing for deep learning frameworks. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 486–498, 2020.

[14] Aaron J. Havens, Alexandre Araujo, Huan Zhang, and Bin Hu. Fine-grained Local Sensitivity Analysis of Standard Dot-Product Self-Attention. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pages 17680–17696. PMLR, 2024.

[15] Nicholas J Higham and Theo Mary. A new approach to probabilistic rounding error analysis. *SIAM journal on scientific computing*, 41(5):A2815–A2835, 2019.

[16] Nicholas J. Higham and Theo Mary. A new approach to probabilistic rounding error analysis. *SIAM Journal on Scientific Computing*, 41(5):A2815–A2835, 2019.

[17] Ilse C. F. Ipsen and Hua Zhou. Probabilistic Error Analysis for Inner Products. *SIAM Journal on Matrix Analysis and Applications*, 41(4):1726–1741, 2020.

[18] Ilse CF Ipsen and Hua Zhou. Probabilistic error analysis for inner products. *SIAM journal on matrix analysis and applications*, 41(4):1726–1741, 2020.

[19] Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. TASO: optimizing deep learning computation with automatic generation of graph substitutions. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 47–62, Huntsville Ontario Canada, October 2019. ACM.

[20] Yuxuan Jiang, Ziming Zhou, Boyu Xu, Beijie Liu, Runhui Xu, and Peng Huang. Training with confidence: catching silent errors in deep learning training with automated proactive checks. In *Proceedings of the 19th USENIX Conference on Operating Systems Design and Implementation*, OSDI '25, USA, 2025. USENIX Association.

[21] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, et al. MegaScale: Scaling large language model training to more than 10,000 GPUs. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 745–760, 2024.

[22] Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, et al. A study of bfloat16 for deep learning training. *arXiv preprint arXiv:1905.12322*, 2019.

[23] Hyunjik Kim, George Papamakarios, and Andriy Mnih. The Lipschitz Constant of Self-Attention. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 5562–5571. PMLR, 2021.

[24] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. Pytorch distributed: Experiences on accelerating data parallel training. *Proceedings of the VLDB Endowment*, 13(12).

[25] Kuiliang Lin, Xiangpu Song, Yingpei Zeng, and Shanqing Guo. DeepDiffer: Find Deep Learning Compiler Bugs via Priority-guided Differential Fuzzing. In *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS)*, pages 616–627, October 2023. ISSN: 2693-9177.

[26] Zhiqi Lin, Youshan Miao, Quanlu Zhang, Fan Yang, Yi Zhu, Cheng Li, Saeed Maleki, Xu Cao, Ning Shang, Yilei Yang, et al. nnScaler: Constraint-guided parallelization plan generation for deep learning training. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 347–363, 2024.

[27] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[28] Amanda Liu, Gilbert Bernstein, Adam Chlipala, and Jonathan Ragan-Kelley. A Verified Compiler for a Functional Tensor Language. *A Verified Compiler for a Functional Tensor Language*, 8(PLDI):160:320–160:342, June 2024.

[29] Jiawei Liu, Jinkun Lin, Fabian Ruffy, Cheng Tan, Jinyang Li, Aurojit Panda, and Lingming Zhang. Nnsmith: Generating diverse and valid test cases for deep learning compilers. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 530–543, 2023.

[30] Jiawei Liu, Yuxiang Wei, Sen Yang, Yinlin Deng, and Lingming Zhang. Coverage-guided tensor compiler fuzzing with joint ir-pass mutation. *Proc. ACM Program. Lang.*, 6(OOPSLA1), April 2022.

[31] Yunchi Lu, Youshan Miao, Cheng Tan, Peng Huang, Yi Zhu, Xian Zhang, and Fan Yang. Trainverify: Equivalence-based verification for distributed llm training. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, pages 237–253, 2025.

[32] Haoyang Ma. A Survey of Modern Compiler Fuzzing, June 2023. arXiv:2306.06884 [cs].

[33] Haoyang Ma, Qingchao Shen, Yongqiang Tian, Junjie Chen, and Shing-Chi Cheung. Fuzzing Deep Learning Compilers with HirGen. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2023, pages 248–260, New York, NY, USA, July 2023. Association for Computing Machinery.

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

[35] Microsoft. Megatron deepspeed. https://github.com/deepspeedai/Megatron-DeepSpeed, 2023.

[36] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM symposium on operating systems principles*, pages 1–15, 2019.

[37] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.

[38] NVIDIA. [bug] switchmlp router weights are not synchronized within the tensor parallelism group. https://github.com/NVIDIA/Megatron-LM/issues/599, 2023.

[39] NVIDIA. [bug] weight gradients with transformerengine v2.1 don't match those with transformerengine v1.12. https://github.com/NVIDIA/TransformerEngine/issues/1616, 2025.

[40] NVIDIA. [bug] wrong attention gradient when context parallelism is on. https://github.com/NVIDIA/TransformerEngine/issues/1557, 2025.

[41] NVIDIA. Transformer engine. https://github.com/NVIDIA/TransformerEngine, 2025. Accessed: 2025-04-01.

[42] Hung Viet Pham, Thibaud Lutellier, Weizhen Qi, and Lin Tan. Cradle: Cross-backend validation to detect and localize bugs in deep learning libraries. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1027–1038, 2019.

[43] Xianbiao Qi, Jianan Wang, and Lei Zhang. Understanding Optimization of Deep Learning via Jacobian Matrix and Lipschitz Constant, November 2023. arXiv:2306.09338 [cs].

[44] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[45] Kevin Scaman and Aladin Virmaux. Lipschitz regularity of deep neural networks: analysis and efficient estimation, October 2019. arXiv:1805.10965 [stat].

[46] Qingchao Shen, Haoyang Ma, Junjie Chen, Yongqiang Tian, Shing-Chi Cheung, and Xiang Chen. A comprehensive study of deep learning compiler bugs. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2021, pages 968–980, New York, NY, USA, August 2021. Association for Computing Machinery.

[47] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[48] Alexey Solovyev, Marek S. Baranowski, Ian Briggs, Charles Jacobsen, Zvonimir Rakamarić, and Ganesh Gopalakrishnan. Rigorous Estimation of Floating-Point Round-Off Errors with Symbolic Taylor Expansions. *ACM Trans. Program. Lang. Syst.*, 41(1):2:1–2:39, December 2018.

[49] Florian Tambon, Amin Nikanjam, Le An, Foutse Khomh, and Giuliano Antoniol. Silent bugs in deep learning frameworks: an empirical study of keras and tensorflow. *Empirical Software Engineering*, 29(1):10, 2024.

[50] Haoyu Wang, Junjie Chen, Chuyue Xie, Shuang Liu, Zan Wang, Qingchao Shen, and Yingquan Zhao. MLIRSmith: Random Program Generation for Fuzzing MLIR Compiler Infrastructure. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1555–1566, September 2023. ISSN: 2643-1572.

[51] Jiannan Wang, Hung Viet Pham, Qi Li, Lin Tan, Yu Guo, Adnan Aziz, and Erik Meijer. D3: Differential testing of distributed deep learning with model generation. *IEEE Transactions on Software Engineering*, 51(1):38–52, 2025.

[52] Zan Wang, Ming Yan, Junjie Chen, Shuang Liu, and Dongdi Zhang. Deep learning library testing via effective model generation. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, pages 788–799, New York, NY, USA, 2020. Association for Computing Machinery.

[53] Zhanghan Wang, Ding Ding, Hang Zhu, Haibin Lin, and Aurojit Panda. It takes two to entangle. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '26, page 1022–1039, New York, NY, USA, 2026. Association for Computing Machinery.

[54] Anjiang Wei, Yinlin Deng, Chenyuan Yang, and Lingming Zhang. Free lunch for testing: fuzzing deep-learning libraries from open source. ICSE '22, pages 995–1007, New York, NY, USA, 2022. Association for Computing Machinery.

[55] Mengdi Wu, Xinhao Cheng, Shengyu Liu, Chunan Shi, Jianan Ji, Man Kit Ao, Praveen Velliengiri, Xupeng Miao, Oded Padon, and Zhihao Jia. Mirage: a multi-level superoptimizer for tensor programs. In *Proceedings of the 19th USENIX Conference on Operating Systems Design and Implementation*, OSDI '25, USA, 2025. USENIX Association.

[56] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On Layer Normalization in the Transformer Architecture. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 10524–10533. PMLR, 2020.

[57] Chenyuan Yang, Yinlin Deng, Jiayi Yao, Yuxing Tu, Hanchi Li, and Lingming Zhang. Fuzzing automatic differentiation in deep-learning libraries. In *Proceedings of the 45th International Conference on Software Engineering*, ICSE '23, pages 1174–1186. IEEE Press, 2023.

[58] Greg Yang. Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation. *arXiv preprint arXiv:1902.04760*, 2019.

[59] Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion models: A comprehensive survey of methods and applications. *ACM Computing Surveys*, 56(4):1–39, 2023.

[60] Zihan Yao, Ruoyu Wu, and Tianxiang Gao. Feature learning dynamics in infinite-depth neural networks, 2026. arXiv:2512.21075v3, revised May 2026.

[61] Xiaoyu Zhang, Weipeng Jiang, Chao Shen, Qi Li, Qian Wang, Chenhao Lin, and Xiaohong Guan. Deep learning library testing: Definition, methods and challenges. *ACM Comput. Surv.*, 57(7), March 2025.

[62] Xufan Zhang, Jiawei Liu, Ning Sun, Chunrong Fang, Jia Liu, Jiang Wang, Dong Chai, and Zhenyu Chen. Duo: Differential Fuzzing for Deep Learning Operators. *IEEE Transactions on Reliability*, 70(4):1671–1685, December 2021. Conference Name: IEEE Transactions on Reliability.

[63] Xufan Zhang, Ning Sun, Chunrong Fang, Jiawei Liu, Jia Liu, Dong Chai, Jiang Wang, and Zhenyu Chen. Predoo: precision testing of deep learning operators. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 400–412, Virtual Denmark, July 2021. ACM.

[64] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

[65] Kahfi S Zulkifli, Wenbo Qian, Shaowei Zhu, Yuan Zhou, Zhen Zhang, and Chang Lou. Verifying semantic equivalence of large models with equality saturation. In *Proceedings of the 5th Workshop on Machine Learning and Systems*, pages 82–89, 2025.

A Case Studies of Silent Bugs

A.1 Categorization of Silent Bugs

We refer to the bugs using the ID listed in Table 1. We categorize the detected silent bugs into three types:

- **Wrong Data (WD)**: The input tensor for an operation deviates from what is expected. This includes the input being stale (bug 2); incorrectly scaled (bug 3, 4); or in a mismatched layout (bug 13), etc.
- **Wrong Communication (WC)**: The order or pattern of the collective communication is incorrect, resulting in an erroneous output tensor. Examples include an incorrect communication order (bug 5), misconfigured communication groups (bug 7), or a wrong communication pattern when computation-communication overlapping is enabled (bug 9), etc.
- **Missing Communication (MC):** One or more collective communication operations are omitted, leaving the result tensor in an incorrect state. A representative example is a missing all-reduce, which causes the tensor to remain a partial sum (bug 6, 12).

A.2 Code Illustration of a Silent Bug

```
def forward(self, input_):
  input_mask = ...
  ...
  # Mask the output embedding.
  if self.tensor_parallel_size > 1:
    #------Correct implementation------#
    output_local[input_mask, :] = 0.0
    #-------Buggy implementation-------#
    input_mask = (masked_input < self.voc_beg_idx) | (masked_input >= self.voc_end_idx)
    output_local[input_mask, :] = 0.0
    #------End of Code Comparison------#
  # Reduce across all tensor parallel ranks.
  output = reduce_from_tp_region(output_local)
  return output
```

Listing 1: The root cause for the example silent bug showed in Figure 1.

Listing 1 is the bug 1 in Table 1. It is also the root cause for the bug shown in Figure 1. The bug occurs in tensor parallelism settings where incorrect entries of the embeddings are selected due to a wrong embedding mask. The error occurs in the embedding layer, and it normally cascades through the model. nnTrace uses the module-wise testing to curb the discrepancy to the faulty module, allowing for precise localization.

A.3 Case Studies on New Bugs

In this section, we highlight nnTrace's ability to uncover subtle silent bugs in training frameworks with more case studies on the new bugs discovered.

Bug 12 results in wrong gradients for normalization layers weights. When sequence parallel (SP) is enabled, the normalization layers must conduct an all-reduce across SP ranks to gather the gradients from different sequence partitions. This reduction should occur explicitly after the forward and backward computations and prior to the weight update. However, due to a flag misconfiguration, this all-reduce operation was skipped, leaving the gradient to be incomplete. This bug is hard to find by other methods because it is a manual step outside of the forward and backward computation graph. nnTrace identified it by detecting inconsistencies among ranks when merging the main gradients.

Bug 14 generates erroneous parameter gradients for layer normalization when both tensor parallel (TP) and context parallel (CP) are enabled in FP8 training. The gradients are wrongly scaled by the tensor parallel size due to the use of an improper reduction operator during communication. Note that this bug is discovered during our comprehensive sweep of parallelism configurations, and it is

not present when using TP or CP alone, or during BF16 training. This highlights the potential of nnTrace to serve as a component in the automated testing pipeline to catch bugs that occur only with specific configurations.

B System Design and Implementation

B.1 Canonical Mapping

To compare tensors in the candidate and reference implementations, it is essential to establish a rigorous correspondence between the tensors. The notion of a *logical full tensor* in the candidate (distributed) runs is useful for the discussion. Such a tensor is assembled from multiple physical shards on multiple devices, and is expected to match a physical tensor in the non-sharded reference trace (see Figure 7). We therefore need to solve two problems with the mapping: 1) how to uniquely identify a tensor in the reference trace or a logical full tensor in the candidate trace; and 2) how to map the local tensor shards in the candidate trace to their corresponding logical full tensors.

For the first problem, we introduce a *canonical identifier* to uniquely identify tensors under non-sharding parallelism strategies, such as DP and PP, distinguishing logical full tensors that are physically separated by data batches or pipeline stages. This identifier is a composite key consisting of the iteration number, micro-batch index, tensor type, and the canonical module name. Each component targets a specific parallel dimension. Specifically, the micro-batch index aligns parallel data batches in the candidate under DP with sequential gradient accumulation steps in the reference. Meanwhile, the canonical module name handles PP by mapping stage-local layer indices (which typically reset to 0) back to global layer indices in the model, as illustrated in Figure 6.

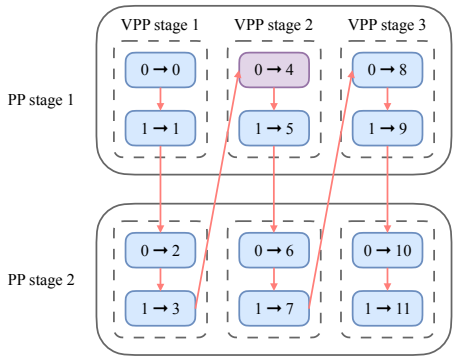

Figure 6: Illustration of the layer index mapping in pipeline parallelism and virtual pipeline parallelism for canonical module name. Each colored box is one layer. The purple example means layer 0 in the 2nd virtual pipeline of the 1st pipeline stage maps to layer 4 in the reference.

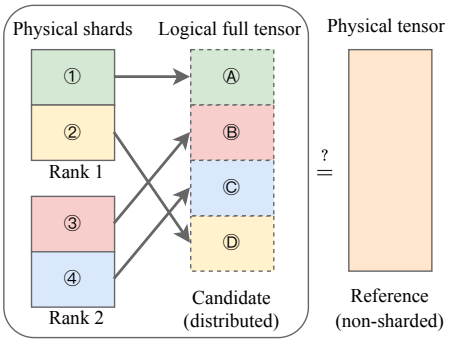

Figure 7: Illustration for canonical shard mapping.

For the second problem, we maintain a data structure called *canonical shard mapping* to handle the parallelism strategies that shard the tensors. It record the mapping from the slices in local shards to those in the corresponding logical full tensor, as shown in Figure 7. A tensor shard on one device can consist of multiple non-contiguous slices in the logical full tensor, e.g., striped attention in context parallelism. In this example, we use the shards from the two ranks to compose the full tensor according to the mapping, and then compare with the physical full tensor from the reference run. When merging the shards into the full tensor, nnTrace will check to ensure that there is no overlap or omission.

Unlike auto-parallelization systems including nnScaler [26] where the tensor mapping are generated when making the parallelization plan, common manually parallelized frameworks like Megatron-LM lacks such mapping information. nnTrace uses the user-provided tensor annotations to infer the canonical shard mapping. For example, in Listing 2, the annotation for the parameters and the input or output activations contain the dimensions on which particular parallelism strategies split.

The mapped slice is inferred from the sharding dimension, parallel rank, and the semantic of the framework.

Developers can extend nnTrace with new parallelization strategies by providing a function that maps the parallelization ranks to shard mappings. In practice, this process requires minimal implementation overhead; existing support for parallelism strategies (e.g., PP, TP, SP, CP) typically comprises approximately 30 lines of code each.

For large models where a single-device reference implementation is not available, we run the reference implementation with pipeline parallelism. Similar to the candidate implementation, the canonical mappings can be established for the reference to facilitate precise comparison with the candidate. More details regarding the rationale and scalability of the reference implementation can be found in Appendix C.4.

```yaml
# parameter annotations
word_embeddings.weight:
  tp_dim: 0

linear_qkv.weight:
  tp_dim: 0

linear_proj.weight:
  tp_dim: 1
...
```

```yaml
# module annotations
linear_qkv:
  input:
    sp_dim: 0
    cp_dim: 0

  output:
    tp_dim: -1
    cp_dim: 0
...
```

Listing 2: Example annotation clips for a GPT model. Module annotations specify which input/output tensors to trace. The numbers specify the tensor dimensions being partitioned by the particular parallelism.

B.2 Consistent Tensor Generator

As demonstrated in Section 3, we need to generate the *same* tensor as input to the candidate and reference modules in order to compare their behavior to identify bugs, as well as when localizing bugs in candidates with the rewriting functionality. These inputs include the model parameters, the input data to the neural networks, and the rewritten intermediate tensors. If we randomly generate a tensor for the reference, we have to ensure that the same logical full tensor is generated for the candidate, with each rank holding its corresponding shard. To achieve this, we use the hash of the tensor's canonical identifier as the seed for the random number generator. This produces the same tensor for the reference implementation and the corresponding logical full tensor for the candidate. The actual distributed tensors supplied to the candidate are then extracted from the generated logical full tensor as slices or shards, based on the shape and slice information from the canonical mapping in Appendix B.1. In addition to generating consistent input tensors to the modules, this mechanism can also be used to generate consistent main gradients to examine the optimizer behavior in the candidate and reference implementations.

B.3 Trace Collector and Tensor Rewrites

The trace collector interfaces with the training framework and the underlying deep learning framework to intercept and record tensor values for comparison. We prioritize a non-intrusive design with minimal user code modifications leveraging PyTorch's native APIs: activations and their gradients are collected through module hooks; parameter gradients are collected through tensor hooks; the main gradients (high-precision accumulator in mixed-precision training) and parameters are collected before and after the optimizer step.

Beyond raw tensor values, canonical identifiers, and shard mappings, we also record auxiliary metadata in the trace, such as the name and class of the module producing the tensor, to facilitate the checking and bug localization process. The traces are stored in the main memory and persisted to disk when the training ends.

As introduced in Step 5 (Section 3), the tensor rewrite functionality extends this infrastructure to enable active bug localization via module-wise testing. It utilizes the same underlying hook-based instrumentation architecture as the trace collector. However, instead of recording tensors, it over-

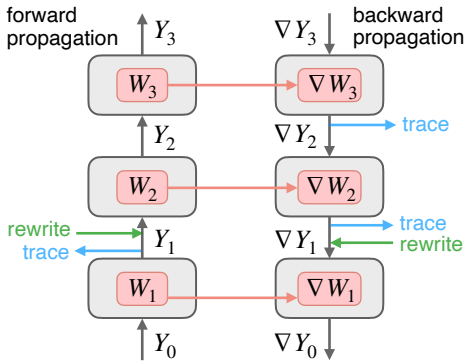

Figure 8: Illustration for trace collector and tensor rewrites in forward and backward propagation

writes the inputs to each module with tensors produced by the consistent distributed tensor generator (Appendix B.2), without altering the computation graph. This ensures that corresponding modules in the reference and candidate implementations consume identical inputs. Consequently, this mechanism isolates the computation of each module, preventing the errors from propagating to downstream computations, thereby enabling precise bug localization.

B.4 Equivalence Checker

After capturing the traces from both the candidate and reference executions, nnTrace uses the equivalence checker to determine whether the two traces are equivalent. The process involves two stages. First, it aggregates the distributed tensor shards to reconstruct the corresponding logical full tensors, enabling direct comparison with the reference tensors. Second, it performs differential testing on the merged tensors and compares the error between the reference and candidate against the dynamic numerical error tolerances obtained in Algorithm 1.

Tensor merger and consistency check. The merger utilizes the canonical shard mapping (Appendix B.1) stored in the traces to reconstruct the logical full tensors from the distributed tensor shards. Crucially, this process also enforces consistency among the tensor shards and reports potential bugs. For example, tensor shards that are replicated across devices must hold equivalent values, e.g., in standard data parallel, the main gradients from each rank must be same. If the merger detects a discrepancy among replicas, such as divergent gradient caused by a missing `all-reduce`, it immediately flags the specific tensor and reports the bug with its location.

Differential testing. Given the merged traces from the reference and the candidate, the equivalence checker performs differential testing on them. First, it computes the differences between the reference and the candidate. It then compares the differences against the expected numerical error tolerance obtained in step 2 in Section 3. If the difference exceeds the tolerance threshold, the checker identifies the deviation as a bug, and localizes the bug by the location of the particular tensor where the violation occurred.

B.5 Large reference models

If the code path of the candidate implementation is independent of the tensor sizes in the training framework code, correctness of such framework code can be checked with a down-scaled model against a single-device reference implementation. In our evaluation, all bugs we discovered in Table 1 can be reproduced and diagnosed using a down-scaled model whose reference implementation fits on a single device. However, there are cases where the framework code path may vary depending on the model size. nnTrace provides a way to run the reference implementation through pipeline parallelism (PP) to support reference models that do not fit in the memory of a single device.

We adopt PP because it is less error-prone and its semantics remain the same as a single-device execution at every pipeline stage. Specifically, our PP implementation employs the pipelining functionality provided natively by PyTorch, without additional system optimizations such as activation checkpointing or computation–communication overlap, thereby minimizing potential sources of im-

plementation bugs. Moreover, the implementation follows the same code path regardless of model size or tensor dimensions, and we validated this PP implementation by cross-checking its outputs with those of a single-device reference on smaller models. A further advantage of PP is that it preserves the unsharded per-layer numerical semantics of single-device execution, since each module fits within a single GPU's memory and no operator partitioning or computation reordering is introduced.

C Additional Experiments

C.1 Experimental Setup

We evaluate nnTrace targeting Megatron-LM and TransformerEngine. The bug reproduction (§5.1) and the efficiency measurements (Appendix C.4) use CUDA 12.4, cuDNN 9.8.0, PyTorch 2.4.1, Megatron-LM 0.11.0, and TransformerEngine 1.13.0 on an Amazon EC2 p5en.48xlarge machine with 8 NVIDIA H200 GPUs; the bug reproductions use the safety margin $k = 2$. The numerical-error experiments (§5.2, §5.3, and Appendices C.2 and C.3) use Megatron-LM at commit `1dcf0da` (Megatron Core 0.16.0rc0), TransformerEngine 2.10.0, PyTorch 2.11.0, CUDA 12.9.1, cuDNN 9.17.1, and NCCL 2.28.9 on an Amazon EC2 p5.48xlarge machine with 8 NVIDIA H100 80GB GPUs.

We tested GPT, Llama, Mamba, and MoE language models. The numerical-error experiments trace one training iteration with four microbatches of four sequences of 1024 tokens from WikiText-103, tokenized with the GPT-2 tokenizer. The reference runs without model parallelism; the candidates use tensor parallelism of size 2 or 4 or context parallelism of size 2. Parameters are initialized as in Assumption 1, except that the state-space parameters of Mamba use the standard Mamba initialization, and are stored in BF16. Every run pads the vocabulary to the same size, so that the reference and the candidates compute the same model. The MoE models use eight experts with top-2 routing, and the router input is rewritten with the reference's value (§3).

Table 2 uses a 24-layer GPT and 32-layer Llama, MoE, and hybrid Mamba models of width 1024 in BF16 and FP8, with a tensor-parallel candidate of size 2 and 12 injected faults per configuration. A single-layer fault perturbs the first decoder layer in the forward pass or the last in backpropagation, and a fault is detected when any tensor it can affect is flagged.

FP8 configurations use TransformerEngine's delayed scaling with the E4M3 format for all FP8 tensors, an amax history of one pass, and amax reduction over the parallel group. Delayed scaling starts from a scaling factor of one, so the first microbatches of an iteration are quantized with factors that do not yet reflect the tensors (Appendix E.7). The per-layer curves therefore show the last microbatch, the statistics of FP8 runs exclude the first, and parameter gradients are accumulated over the iteration.

C.2 Justification for the Safety Margin

The safety margin k in Algorithm 1 separates the ratios e_T/τ_T of correct candidates from those of faulty ones. Figure 9 shows their distribution over the bug-free tensor-parallel runs. No correct-run ratio exceeds 2.9, including those of the context-parallel candidates and of repeated runs with other perturbation and initialization seeds, whereas every detected fault of Table 2 has a ratio above 5. Any k in this gap gives the same results; we use $k = 3.5$. The margin also holds out of sample: choosing k as 1.25 times the largest ratio of one part of the configurations, split by width, depth, architecture, or precision, gives values between 2.8 and 3.6, and $k = 3.5$ raises no false alarm on any held-out part.

The comparison with `torch.allclose` in Table 2 changes both the threshold and the metric. Fixed thresholds on the relative error e_T that nnTrace uses, set from the correct runs of the other configurations, still miss 7 to 26 of the 96 faults, depending on whether they are set per tensor type and precision or globally, and they raise false alarms.

The injected faults have the form of the proxy's perturbations, only larger, and do not reproduce real bugs; this calibration describes the tested configurations and fault model.

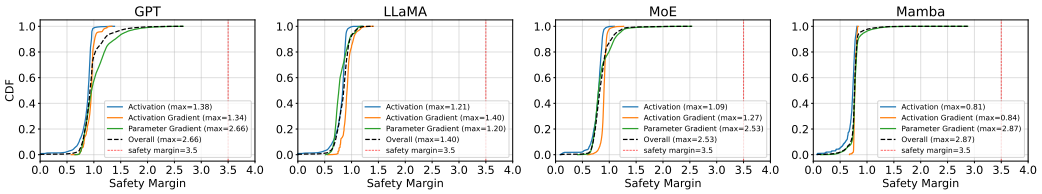

Figure 9: Cumulative distribution of the ratio e_T/τ_T between observed and estimated numerical error over the traced tensors of the bug-free tensor-parallel runs, by architecture and tensor type. The dashed red line marks the safety margin $k = 3.5$.

C.3 Additional Numerical Error Estimation Results

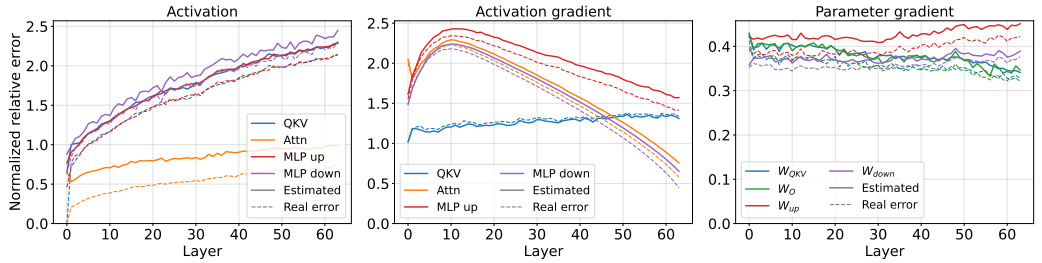

Figure 10: Per-module estimation accuracy for a 64-layer GPT (BF16, $d = 1024$) configuration. The observed numerical error is from tensor parallelism of size 2. The y-axis is normalized by the machine epsilon of BF16.

Figure 10 shows the per-module error tolerance estimation and the real numerical error from a correctly implemented tensor parallelism. Similar to the Llama results in Section 5.2, the estimation accurately captures the errors of different modules and tensor types.

Figures 11 and 12 show the full comparison of our numerical error estimation with the real numerical error from correctly implemented tensor parallelism and context parallelism, respectively. They are evaluated across combinations of model architectures (GPT, Llama, Mamba, MoE), numerical precisions (BF16, FP8), and layer counts. Each subplot overlays the estimated numerical error tolerance (solid lines, d=1024) against the available observations from the sweep over tensor-parallel sizes 2 and 4 or context-parallel size 2 and hidden dimensions $d \in \{512, 1024\}$. Not every panel contains every size and width combination. Colors distinguish activations, activation gradients, and parameter gradients. In the per-layer curves of Figures 4, 11 and 12, the activation is the output of a decoder layer and the activation gradient is the gradient with respect to that output; the parameter gradient is that of the QKV projection weight for GPT and Llama, of the input projection of the Mamba mixer for Mamba (whose attention and MLP layers have no point), and, for MoE, the mean of the relative errors of the eight experts' first FFN weights. All y-axes report relative error normalized by the machine epsilon of BF16.

Across the 32 evaluated configurations, the full sweep shows the same qualitative agreement between estimated and observed errors as the representative cases in §5.2. Mamba FP8 models have no tensor-parallel candidate at width 512 and none of size 4 at width 1024, because their sharded projections do not meet the alignment requirements of TransformerEngine's FP8 kernels.

The largest ratios in Figure 9 occur in a few parameter gradients under tensor parallelism of size 4: bias gradients in GPT and MoE, and one state-space parameter in Mamba. A bias gradient is a token mean of gradient rows, so a rounding-error component common to many tokens enters it without averaging (Appendix E.6). These ratios stay below the safety margin.

C.4 System Efficiency and Scalability

Table 3 shows the tracing times for the pipelined references of models that do not fit on a single device. The overheads mostly come from the device to host memory movement. Notice that the

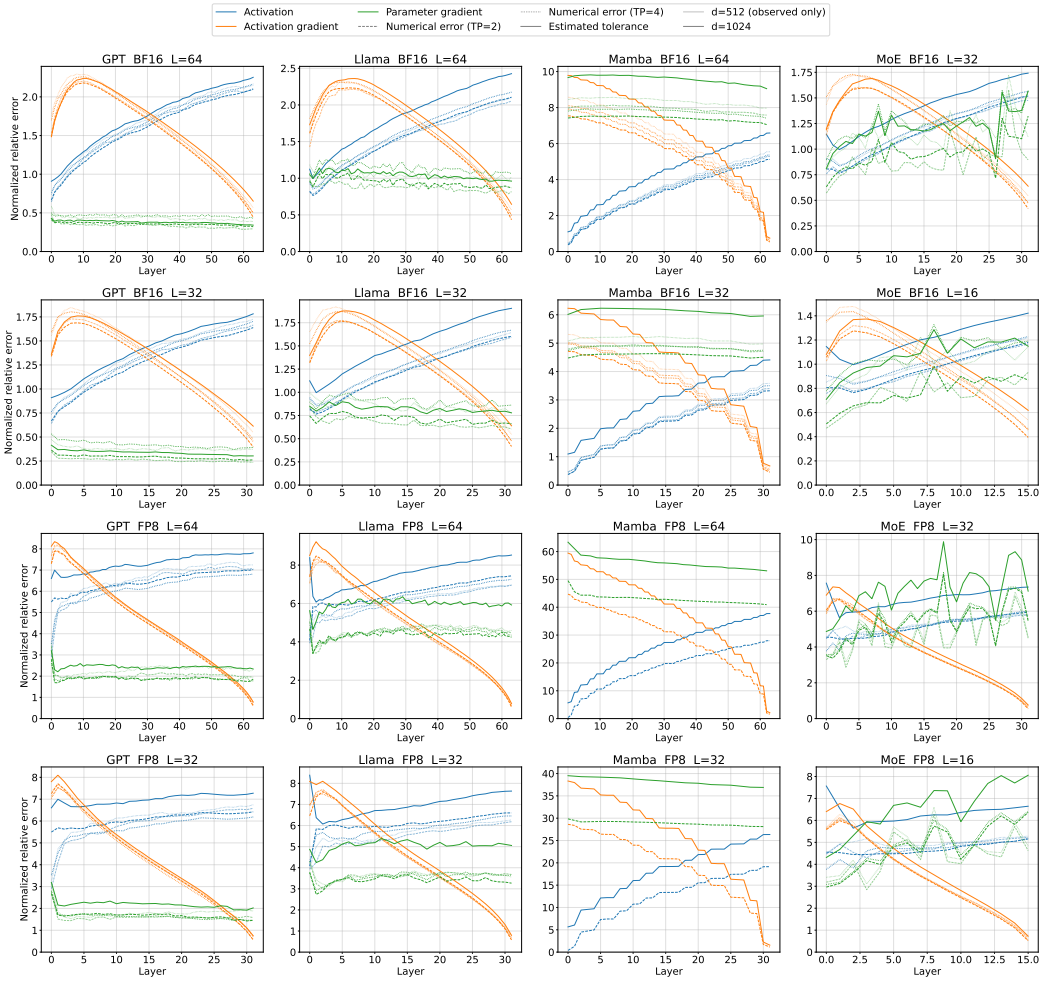

Figure 11: Additional estimated numerical error versus observed numerical error from tensor parallel runs with a sweep over architecture, precision, and configuration. The y-axis is the relative error normalized by the machine epsilon of BF16. FP8 curves use the same normalization (Appendix E.7).

Table 3: The time for nnTrace reference execution on models beyond single-device capacity.

Model class	Llama 3			DeepSeek V3		
Model size	8B	70B	405B	16B	236B	671B
Time (mins)	1.8	7.2	17.1	2.8	15.2	22.7

candidate implementations usually run faster than the reference implementations due to optimizations or more parallelisms enabled. Even for the DeepSeek V3 671B, one of the largest open source language models, the tracing time of nnTrace remains reasonable.

D Discussions on Verification-Based Methods

The difficulty of debugging for distributed training has motivated several recent verification-based solutions [31, 65, 53]. These works, which are concurrent with our own, aim to prove semantic equivalence of the computation graphs generated in the distributed and single-device settings. One major benefit of these works is that they can produce a proof artifact that formally establishes the correctness of the computation graph if verification succeeds. Nevertheless, there are challenges in adopting these methods in production.

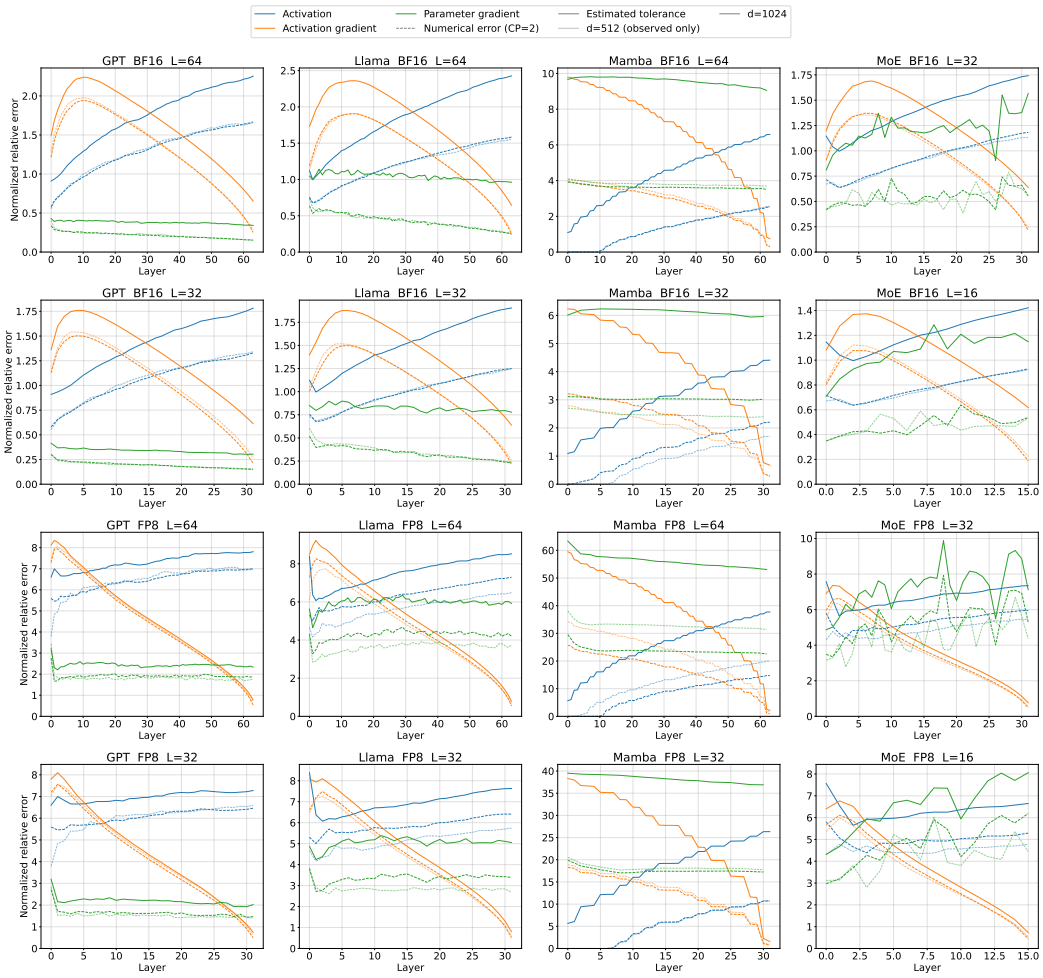

Figure 12: Additional estimated numerical error versus observed numerical error from context parallel runs with a sweep over architecture, precision, and configuration. The y-axis is the relative error normalized by the machine epsilon of BF16. FP8 curves use the same normalization (Appendix E.7).

TrainVerify [31] symbolizes the graph-based execution plans from the auto-parallel frameworks, then converts them into logical formulas, and verifies their mathematical equivalence using SMT solvers. Their workflow has several limitations, which may make it challenging to deploy in production for any distributed training environment. First, implementation and maintenance cost may be high: users of TrainVerify need to implement and maintain a separate code base from the parallelization framework itself to convert the model and its execution plan into a symbolic data flow graph for verification, along with code that interprets symbolic operators by emitting SMT formulas that accurately describe the operator semantics given all possible combinations of parameters; additionally many distributed frameworks (e.g., Megatron-LM) do not provide out-of-the-box the lineage metadata required by TrainVerify to establish the proof targets. Secondly, the "parallelization equivalence" established by TrainVerify on the computation graphs may not actually guarantee desired behavior in training, since it does not rule out bugs that exist on a lower-level than the computation graphs, such as misuse of custom kernels [40], or issues due to numerical instability of mathematically equivalence expressions since the verification is based on real variables. Finally, localizing issues to specific components may be challenging if the SMT solver based approach fails to prove the equivalence of two expressions.

In addition to the above challenges, framework users and maintainers may also be hesitant to adopt verification-based methods due to scalability concerns–both SMT solvers and e-graph based methods might suffer from exponential growth in the running time in the worst case, as size of SMT formulas or number of rules grow.

By contrast, differential testing is intuitive for the developers, and it has several advantages compared to these verification-based solutions. First, under differential testing, the framework code and kernels being executed in the debugging process is usually the same as what will be executed during the actual training process. This can save the developers from efforts in writing and maintaining separate specification required for verification. Additionally, executing the actual code directly can potentially expose GPU kernel-related issues and other low-level problems that are abstracted away in order for verification to be tractable. Second, once a bug is successfully reproduced and fixed by differential testing, the developers can usually transform their debugging code to a test in an automated pipeline to prevent regression.

E An Analysis of Floating-Point Differences in Distributed Transformer Training

The error estimates in §4 depend on how numerical differences are introduced locally and how the surrounding computation propagates them. We develop these two aspects for a randomly initialized transformer with pre-layer normalization (Pre-LN), beginning with the growth of the residual stream's second moment and its effect on LayerNorm's local Lipschitz constant. These properties lead to bounds for activations and gradients under explicit assumptions on local rounding errors and their amplification.

The accumulation bounds explain the depth dependence of numerical errors, while their constants still depend on the operations used to compute each tensor. To connect the analysis to the tolerance rule, we then compare two complete runs of the computation and show how the reference perturbation response can estimate the candidate–reference difference. Throughout, we distinguish properties supplied by the initialization from statistical conditions on the rounding errors; the latter are stated where they enter the proofs and discussed together at the end.

E.1 Model and notation

Let $X \in \mathbb{R}^{n\times d}$ contain the input vectors at n token positions, counting positions across the batch, and let d be the residual width. We compare the real-arithmetic computation $F = f_L \circ \cdots \circ f_1$ with a floating-point implementation $\hat{F}$, starting from the same input:

$$Y_0 = \hat{Y}_0 = X, \qquad Y_l = f_l(Y_{l-1}), \qquad \hat{Y}_l = \hat{f}_l(\hat{Y}_{l-1}).$$

All tensor norms and inner products are Frobenius, or equivalently Euclidean after vectorization; $\|J\|_2$ denotes the spectral norm, induced by the Euclidean norm. We write $u = \varepsilon_{\mathrm{mch}}/2$ for unit roundoff, where machine epsilon $\varepsilon_{\mathrm{mch}}$ is the spacing from one to the next representable number. For an ideal tensor T and its computed counterpart $\hat{T}$, the root mean square (RMS) relative error is

$$\mathcal{E}(\hat{T}, T) = \frac{\sqrt{\mathbb{E}\|\hat{T} - T\|^2}}{\sqrt{\mathbb{E}\|T\|^2}}.$$

Expectations include initialization, inputs, and the probabilistic rounding model, and are taken before normalization. In that model, local rounding errors are random variables whose only assumed properties are the moment conditions stated where they are used. For BF16, $\varepsilon_{\mathrm{BF16}} = 2^{-7}$ and $u_{\mathrm{BF16}} = 2^{-8}$. All moments used below are finite, and relative-error denominators are nonzero. Constants may depend on the architecture and precision and reduction scheme. Except for amplification factors that carry a layer index, they are uniform in the layer index and depth range under analysis.

We consider a Pre-LN GPT block with an attention branch $\mathcal{A}_l$ and a feed-forward network (FFN) branch $\mathcal{M}_l$:

$$Z_l = Y_{l-1} + \mathcal{A}_l(Y_{l-1}), \qquad Y_l = Z_l + \mathcal{M}_l(Z_l).$$

Each branch begins with LayerNorm and ends with its own linear output projection and bias. The FFN uses GELU and has hidden width proportional to d, while attention splits the width into heads of widths d_h with $\sum_h d_h = d$ and uses a fixed mask with at least one unmasked entry in each row. LayerNorm includes gain γ, bias β, and a positive numerical stabilizer ϵ_{LN}. We write $P = I - \mathbf{1}\mathbf{1}^\top/d$ for the orthogonal projection that subtracts the coordinate mean, with $\mathbf{1}$ denoting the all-ones vector, and $s(x)^2 = \|Px\|^2/d + \epsilon_{\mathrm{LN}}$, so that $\mathrm{LayerNorm}(x) = \gamma \odot Px/s(x) + \beta$. This is the residual structure studied in [56], with parameter scales chosen to match the implementation.

Assumption 1 (Random initialization)**.** *Before storage rounding, each weight entry is drawn independently from* $\mathcal{U}[-1/\sqrt{\text{fan_in}}, 1/\sqrt{\text{fan_in}}]$*, and each LayerNorm gain entry is drawn independently from* $\mathcal{U}[1 - 1/\sqrt{d}, 1 + 1/\sqrt{d}]$*. All biases, of LayerNorm and of the linear maps, are zero. Here* fan_in *is the input width of the corresponding linear map. Distinct parameter tensors are initialized independently; a tied input/output embedding matrix is drawn once. The input is the sum of word and positional embeddings with entries drawn from* $\mathcal{U}[-1/\sqrt{d}, 1/\sqrt{d}]$*, with token indices independent of the initialization.*

The distributional statements describe the ideal counterparts of the initialized parameters; storage casts within the layers and arithmetic errors enter the local rounding terms below. Although distinct branch parameters are independent, vectors at different sequence positions share parameters and may be strongly correlated. The arguments therefore allow dependent positions and dependent inputs and target labels.

Gains near one and zero biases are close to the common initialization with unit gains. Conditional on its input x, a LayerNorm output then has expected squared norm $(1 + \frac{1}{3d})\|Px\|^2/s(x)^2 \le d + \frac{1}{3}$ (Lemma 5), which is close to d when the stabilizer is negligible, the usual scale of normalized transformer activations. The scale of the logits also depends on the output head; small gains would make the logits small and the output distribution nearly uniform, and Appendix E.6 explains why this matters for parameter gradients.

E.2 Residual second moments and LayerNorm sensitivity

Relative error depends on the size of the ideal activation as well as that of the accumulated discrepancy. We first bound its expected squared norm using the zero-mean weights of each residual branch's output projection. Conditional on the branch input and its internal computation, the branch output has mean zero, so its expected inner product with the incoming state vanishes. The same observation will later control the amplification of a difference between two inputs.

Lemma 4 (Second-moment growth of the residual stream)**.** *Under Assumption 1, let* $d \ge 2$*, let the FFN width be* $4d$*, and, for a LayerNorm whose input at token* i *is* x_i*, write* $z_i = Px_i/s(x_i)$ *for its normalized input. Suppose that every FFN LayerNorm satisfies the* signal condition*, a lower bound on the energy of its normalized inputs that holds when the stabilizer* ϵ_{LN} *is negligible on average,*

$$\sum_{i=1}^{n} \mathbb{E}\|z_i\|^2 \ge \kappa\, nd$$

for some $\kappa \in (0, 1]$*. Then* $N_0 = 2n/3$ *and, for* $l \ge 1$*,*

$$N_0 + \lambda\, nl \le N_l := \mathbb{E}\|Y_l\|_F^2 \le N_0 + \Lambda\, nl,$$

with

$$\lambda = \frac{\kappa d}{36}, \qquad \Lambda = \frac{(\sqrt{d}+1)^2 + d + 1/3}{9}.$$

Consequently $c_N n(l+1) \le N_l$ *for* $l \ge 1$ *and* $N_l \le C_N n(l+1)$ *for* $l \ge 0$*, with* $c_N = \lambda/2$ *and* $C_N = \Lambda$*. The ratio*

$$\frac{C_N}{c_N} = \frac{16}{\kappa}\Big(1 + \frac{1}{\sqrt{d}} + \frac{2}{3d}\Big)$$

is bounded uniformly in d *when* κ *is bounded below. Moreover,* $\mathbb{E}\|Y_l P\|_F^2 = (1 - 1/d)N_l$*.*

Proof. Consider either residual branch, and write its output as

$$R = HW, \qquad H \in \mathbb{R}^{n\times r}, \quad W \in \mathbb{R}^{r\times d}.$$

Let $\mathscr{H}$ be the σ-algebra generated by the branch input and all parameters used to form H. The output projection W is independent of this information, and its entries have mean zero and variance $1/(3r)$. Averaging over it gives

$$\mathbb{E}[R \mid \mathscr{H}] = 0, \qquad \mathbb{E}[\|R\|_F^2 \mid \mathscr{H}] = \frac{d}{3r}\|H\|_F^2.$$

The first identity implies that the cross term between R and its branch input has zero expectation. Applying it successively to the attention and FFN residual additions yields

$$N_l = N_{l-1} + \mathbb{E}\|\mathcal{A}_l(Y_{l-1})\|_F^2 + \mathbb{E}\|\mathcal{M}_l(Z_l)\|_F^2,$$

so it suffices to bound the two branch terms.

For the upper bound, let o_j be the output of the attention LayerNorm at token j. Since $\|Px/s(x)\|^2 \le d$ and $|\gamma_k| \le 1 + d^{-1/2}$, every LayerNorm output satisfies $\|o_j\| \le \|\gamma\|_\infty \sqrt{d} \le \sqrt{d} + 1$. Condition on the LayerNorm outputs and on the query and key projections, which determine the attention weights p_{ij}. The value projection $W_V^h \in \mathbb{R}^{d \times d_h}$ of head h is independent of this information, and by linearity the context of token i in that head is $c_i^h = \big(\sum_j p_{ij} o_j\big) W_V^h$. Because the weights are nonnegative and sum to one, $\|\sum_j p_{ij} o_j\| \le \max_j \|o_j\| \le \sqrt{d} + 1$, so

$$\mathbb{E}\big[\|c_i^h\|^2 \mid o, p\big] = \frac{d_h}{3d}\Big\|\sum_j p_{ij} o_j\Big\|^2 \le \frac{d_h}{3d}(\sqrt{d} + 1)^2.$$

Summing over heads, with $\sum_h d_h = d$, and applying the output projection bounds the attention branch by $(\sqrt{d} + 1)^2/9$ per token. For the FFN, Lemma 5 bounds the conditional second moment of the LayerNorm output by $d + 1/3$ per token, the first projection multiplies it by $4/3$, GELU does not increase it because $|\mathrm{GELU}(t)| \le |t|$, and the second projection divides it by twelve. The two branches together add at most Λ per token.

For the lower bound, we use the FFN alone. Condition on its LayerNorm output $v = \mathrm{LayerNorm}(Z_l)$, and write the pre-activation of hidden unit j at token i as $t_{ij} = \langle v_i, w_j \rangle$, where w_j is the corresponding column of the first projection. The columns w_j are independent of v and have symmetric distributions, so t_{ij} is conditionally symmetric. For the standard normal distribution function Φ, $\Phi(t) + \Phi(-t) = 1$, hence

$$\mathrm{GELU}(t)^2 + \mathrm{GELU}(-t)^2 = t^2\big(\Phi(t)^2 + \Phi(-t)^2\big) \ge \frac{t^2}{2},$$

and symmetry gives $\mathbb{E}[\mathrm{GELU}(t_{ij})^2 \mid v] \ge \frac{1}{4}\mathbb{E}[t_{ij}^2 \mid v]$. The same holds for the tanh approximation of GELU, which has the form $t\, g(t)$ with $g(t) + g(-t) = 1$. Summing over the $4d$ hidden units and applying the output projection,

$$\mathbb{E}[\|\mathcal{M}_l(Z_l)_i\|^2 \mid v] \ge \frac{1}{12} \cdot \frac{1}{4} \cdot \frac{4}{3}\|v_i\|^2 = \frac{\|v_i\|^2}{36}.$$

Taking expectations and applying Lemma 5 conditionally on Z_l gives $\mathbb{E}\|\mathcal{M}_l(Z_l)_i\|^2 \ge \frac{1}{36}\mathbb{E}\|v_i\|^2 \ge \frac{1}{36}\mathbb{E}\|z_i\|^2$, where z_i is the normalized input of the FFN LayerNorm. Summing over tokens and using the signal condition, the FFN adds at least λn. The embedding gives $N_0 = 2n/3$, and summing the increments proves the two-sided bound. For $l \ge 1$, $N_0 + \lambda n l \ge \lambda n l \ge \frac{\lambda}{2} n(l+1)$, and $N_0 \le \Lambda n$ for $d \ge 2$ gives the upper bound in the stated form.

To check centering, regard a token's branch output R_i as a column vector. Since the output projection is independent and isotropic,

$$\mathbb{E}[R_i R_i^\top \mid \mathscr{H}] = \frac{\|H_{i,:}\|^2}{3r} I_d.$$

The initial token's second-moment matrix is also proportional to the identity, and the residual cross moments vanish because the branch output has conditional mean zero. Induction therefore gives $\mathbb{E}[Y_{l,i} Y_{l,i}^\top] = q_{l,i} I_d$ for some $q_{l,i} \ge 0$. Since P is an orthogonal projection of rank $d - 1$,

$$\mathbb{E}\|PY_{l,i}\|^2 = (d-1) q_{l,i} = (1 - 1/d)\mathbb{E}\|Y_{l,i}\|^2,$$

and summing over tokens proves the last claim. □

Since $\|z\|^2/d = \|Px\|^2/(\|Px\|^2 + d\epsilon_{\mathrm{LN}})$, the signal condition requires the centered LayerNorm inputs to be large compared with $d\epsilon_{\mathrm{LN}}$ on average; a pointwise bound $\|Px_i\|^2 \ge c\, d\epsilon_{\mathrm{LN}}$ for all tokens gives $\kappa = c/(1 + c)$. Appendix E.8 evaluates κ and the resulting constants for the implemented model.

The lemma bounds an expectation over initialization, whereas the sensitivity of an individual LayerNorm operation depends on its own normalization denominator. For a LayerNorm with input

$X \in \mathbb{R}^{n\times d}$, write $\bar{s}^2 = \mathbb{E}\|X\|_F^2/(nd)$ for the expected per-coordinate second moment of its input. For both LayerNorms of layer l, whose inputs are Y_{l-1} and Z_l, Lemma 4 gives

$$\bar{s}^2 \geq \frac{N_{l-1}}{nd} \geq \frac{2/3 + \lambda(l-1)}{d} = \frac{2}{3d} + \frac{\kappa(l-1)}{36},$$

which grows linearly in depth beyond the first layer. The main text assumes that the denominators near the inputs being compared do not fall far below this expected value. The next lemma states the required lower bound and derives a local Lipschitz bound from it.

Lemma 5 (Local Lipschitz bound for LayerNorm)**.** *Under Assumption 1, let $\bar{s}^2$ be the expected per-coordinate second moment of a LayerNorm's input, and suppose the inputs x in the comparison neighborhood, the union of the segments between the ideal and computed inputs being compared, satisfy*

$$s(x)^2 = \|Px\|^2/d + \epsilon_{\mathrm{LN}} \geq \nu\bar{s}^2$$

for some $\nu > 0$. Then the Jacobian of LayerNorm has spectral norm at most $(1 + d^{-1/2})(\nu\bar{s}^2)^{-1/2}$, giving the same Lipschitz bound along any segment contained in that neighborhood. Independently of the denominator condition, for an input x independent of the LayerNorm's gain, with $z = Px/s(x)$,

$$\mathbb{E}[\|\operatorname{LayerNorm}(x)\|^2 \mid x] = \Big(1 + \frac{1}{3d}\Big)\|z\|^2, \qquad \textit{so} \qquad \|z\|^2 \leq \mathbb{E}[\|\operatorname{LayerNorm}(x)\|^2 \mid x] \leq d + \frac{1}{3}.$$

Proof. Write $y = Px$ and $D_\gamma = \operatorname{diag}(\gamma)$, so that $\operatorname{LayerNorm}(x) = D_\gamma y/s(x) + \beta$. In a direction h, differentiation of the denominator gives

$$\mathrm{d}s(x)[h] = \frac{\langle y, Ph\rangle}{d\, s(x)} = \frac{\langle y, h\rangle}{d\, s(x)},$$

where the second equality uses $Py = y$. Applying the quotient rule therefore yields

$$J_{\operatorname{LayerNorm}}(x) = \frac{D_\gamma}{s(x)}\left(P - \frac{yy^\top}{d\, s(x)^2}\right).$$

The matrix in parentheses vanishes on the constant direction and acts as the identity on centered vectors orthogonal to y. If $y \neq 0$, its eigenvalue in the y direction is

$$1 - \frac{\|y\|^2}{d\, s(x)^2} = \frac{\epsilon_{\mathrm{LN}}}{s(x)^2} \in [0, 1];$$

when $y = 0$, the matrix is simply P. Its operator norm is therefore at most one, and

$$\|J_{\operatorname{LayerNorm}}(x)\|_2 \leq \frac{\|\gamma\|_\infty}{s(x)} \leq \frac{1 + d^{-1/2}}{\sqrt{\nu}\,\bar{s}}.$$

Integrating this derivative along a segment contained in the comparison neighborhood gives the corresponding finite-difference bound.

For the output second moment, condition on x and put $z = Px/s(x)$. The gain entries are independent of x and of each other, with mean one and variance $1/(3d)$, so $\mathbb{E}[\gamma_k^2] = 1 + 1/(3d)$ and, since $\beta = 0$,

$$\mathbb{E}[\|D_\gamma z\|^2 \mid x] = \sum_{k=1}^{d}\Big(1 + \frac{1}{3d}\Big)z_k^2 = \Big(1 + \frac{1}{3d}\Big)\|z\|^2.$$

Since $0 \leq \|z\|^2 \leq d$, this proves the stated bounds. This identity uses independence of the input from the gain; the denominator lower bound is needed only for the Lipschitz estimate. □

In the first layer, the lower bound on $\bar{s}^2$ is $2/(3d)$, the per-coordinate second moment of the embedding, and the Jacobian bound is of order $d^{1/2}$; from the second layer on, it decreases as $l^{-1/2}$. The decreasing Lipschitz bound for LayerNorm explains why a residual branch can have a smaller effect on an incoming discrepancy at greater depth. To use this observation in an error recurrence, we need a bound on the expected squared difference through the complete branch, including its internal computation and fluctuations over initialization. We state this mean-square perturbation bound directly for the input pairs encountered in the comparison.

Assumption 2 (Mean-square bounds on branch perturbations)**.** *For the input pairs U, V encountered at layer l, assume independence from the current layer's parameters and put $U' = U + \mathcal{A}_l(U)$ and $V' = V + \mathcal{A}_l(V)$. Assume*

$$\mathbb{E}\|\mathcal{A}_l(U) - \mathcal{A}_l(V)\|^2 \le \frac{a_l}{l}\mathbb{E}\|U - V\|^2, \qquad \mathbb{E}\|\mathcal{M}_l(U') - \mathcal{M}_l(V')\|^2 \le \frac{b_l}{l}\mathbb{E}\|U' - V'\|^2,$$

with $a_l, b_l \ge 0$, and put $c'_l = a_l + b_l + a_l b_l / l$. There are a threshold layer $l_0 \ge 2$ and a constant $0 \le \bar{c} < 2$, both independent of depth, such that $c'_l \le \bar{c}$ for all $l \ge l_0$; the constants a_l, b_l with $l < l_0$ are also independent of depth.

The two inputs may be dependent on each other and on preceding layers; independence is required only from the current layer's parameters. The constants a_l, b_l bound amplification in mean square within the two branches; the condition $c'_l \le \bar{c} < 2$ from the threshold layer l_0 on is the moderate-amplification condition used when summing forward errors. It is not imposed on the transient layers $l < l_0$, whose constants may be larger; the measurements in Appendix E.8 show a large factor in the first layer, whose input, the embedding, has a much smaller second moment than the branch outputs. The following lemma shows how the branch bounds combine without introducing a cross term of order $l^{-1/2}$.

Lemma 6 (Mean-square layer amplification)**.** *Under Assumptions 1 and 2,*

$$\mathbb{E}\|f_l(U) - f_l(V)\|^2 \le (1 + a_l/l)(1 + b_l/l)\mathbb{E}\|U - V\|^2 = (1 + c'_l/l)\mathbb{E}\|U - V\|^2.$$

Proof. Put $D = U - V$ and $D_{\mathcal{A}} = \mathcal{A}_l(U) - \mathcal{A}_l(V)$. The shared attention output bias cancels in this difference, which has the form

$$D_{\mathcal{A}} = (H_{\mathcal{A}}(U) - H_{\mathcal{A}}(V))W_{\mathcal{A}}.$$

Conditioning on U, V and the internal attention parameters leaves $W_{\mathcal{A}}$ independent and mean zero. Hence $\mathbb{E}\langle D, D_{\mathcal{A}}\rangle = 0$, and expanding the residual addition gives

$$\mathbb{E}\|U' - V'\|^2 = \mathbb{E}\|D\|^2 + \mathbb{E}\|D_{\mathcal{A}}\|^2 \le (1 + a_l/l)\mathbb{E}\|U - V\|^2.$$

For the FFN difference, we condition instead on U', V' and its internal parameters. Its independent, zero-mean output matrix again removes the expected cross term with the incoming difference, so

$$\mathbb{E}\|f_l(U) - f_l(V)\|^2 \le (1 + b_l/l)\mathbb{E}\|U' - V'\|^2.$$

Combining these inequalities gives the claim; the equality in the statement is the expansion $(1 + a_l/l)(1 + b_l/l) = 1 + (a_l + b_l)/l + a_l b_l / l^2$. □

E.3 Forward error accumulation

We now apply the layer-amplification estimate to the difference between the computed and ideal forward states. The useful decomposition evaluates the local rounding error at the computed input, so that the remaining term is exactly a difference through the real-arithmetic layer:

$$\begin{aligned}
\Delta_l &:= \hat{Y}_l - Y_l = \xi_l + \rho_l, \\
\xi_l &= \hat{f}_l(\hat{Y}_{l-1}) - f_l(\hat{Y}_{l-1}), \\
\rho_l &= f_l(\hat{Y}_{l-1}) - f_l(Y_{l-1}).
\end{aligned}$$

This identity separates the error introduced at layer l from the discrepancy inherited from earlier layers without linearizing f_l. To bound its second moment, we need both a local error scale and control of the cross term between these two contributions.

Assumption 3 (Forward rounding moments)**.** *For constants $\eta, c_F \ge 0$ independent of depth,*

$$\mathbb{E}\|\xi_l\|^2 \le \eta^2 u^2 N_l, \qquad \mathbb{E}\langle \xi_l, \rho_l\rangle \le c_F \mathbb{E}\|\xi_l\|^2.$$

The first inequality bounds the local mean-square rounding error relative to the activation's second moment, with η absorbing the operations and precision choices within a layer. The second limits the extent to which a local error reinforces the propagated discrepancy. Conditional mean zero, $\mathbb{E}[\xi_l \mid \rho_l] = 0$, would give $c_F = 0$, but the stated condition also permits nonzero cross moments. Both bounds concern the complete local error on the encountered traces; zero-mean rounding errors at individual operations alone need not establish them.

Theorem 7 (Activation error)**.** *Under Assumptions 1, 2, and 3,*

$$\mathbb{E}\|\Delta_l\|^2 \le (1+2c_F)\eta^2u^2\sum_{j=1}^{l} N_j \prod_{k=j+1}^{l}(1+c'_k/k),$$

where an empty product equals one. Equivalently, dividing by N_l gives the profile bound

$$\mathcal{E}(\hat{Y}_l, Y_l)^2 \le (1+2c_F)\eta^2u^2\sum_{j=1}^{l} \frac{N_j}{N_l} \prod_{k=j+1}^{l}(1+c'_k/k). \tag{1}$$

In particular, put

$$\Gamma = \prod_{k=2}^{l_0-1} \max\Big\{1, \frac{1+c'_k/k}{1+\bar{c}/k}\Big\}.$$

If the hypotheses of Lemma 4 also hold, its second-moment bounds give, for $l \ge 1$, the closed-form bound

$$\mathcal{E}(\hat{Y}_l, Y_l)^2 \le \frac{\Gamma(1+2c_F)C_N}{(2-\bar{c})c_N}\eta^2u^2(l+2), \tag{2}$$

and hence

$$\mathcal{E}(\hat{Y}_l, Y_l) = O(\eta u\sqrt{l}).$$

Proof. Let $V_l = \mathbb{E}\|\Delta_l\|^2$. Expanding the error decomposition and applying Assumption 3 gives

$$\begin{aligned} V_l &= \mathbb{E}\|\rho_l\|^2 + 2\mathbb{E}\langle\xi_l, \rho_l\rangle + \mathbb{E}\|\xi_l\|^2 \\ &\le \mathbb{E}\|\rho_l\|^2 + (1+2c_F)\eta^2u^2N_l. \end{aligned}$$

The pair $(\hat{Y}_{l-1}, Y_{l-1})$ depends on preceding layers and is independent of the current layer's parameters, as required by Assumption 2. Lemma 6 therefore bounds the propagated term by $(1+c'_l/l)V_{l-1}$, yielding

$$V_l \le (1+2c_F)\eta^2u^2N_l + (1+c'_l/l)V_{l-1}.$$

Iterating from $V_0 = 0$ proves the first assertion, and dividing by N_l gives (1): an error introduced at layer j acquires one amplification factor for each subsequent layer. The factor of the first layer never multiplies an error, because $V_0 = 0$.

For the closed-form bound, use $N_l \le C_N n(l+1)$ and put

$$B = (1+2c_F)\eta^2u^2C_Nn, \qquad Q_l = \frac{B}{2-\bar{c}}(l+1)(l+2), \qquad \chi_k = \max\Big\{1, \frac{1+c'_k/k}{1+\bar{c}/k}\Big\}\ (k \ge 2).$$

Then $\chi_k = 1$ for $k \ge l_0$, and $\prod_{k=2}^{l}\chi_k \le \Gamma$. Let $W_1 = V_1$ and $W_l = V_l/\prod_{k=2}^{l}\chi_k$ for $l \ge 2$. Since $1 + c'_l/l \le \chi_l(1+\bar{c}/l)$ and the products are at least one,

$$W_1 \le 2B, \qquad W_l \le (1+\bar{c}/l)W_{l-1} + B(l+1) \quad (l \ge 2).$$

The condition $\bar{c} < 2$ makes Q_l nonnegative, and direct substitution shows that, for $l \ge 1$,

$$(1+\bar{c}/l)Q_{l-1} + B(l+1) = \frac{B}{2-\bar{c}}(l+1)\big[l + \bar{c} + (2-\bar{c})\big] = Q_l.$$

Since $W_1 \le 2B \le Q_1$ and $1 + \bar{c}/l > 0$, induction gives $W_l \le Q_l$, hence $V_l \le \Gamma Q_l$. Dividing by the lower bound $N_l \ge c_N n(l+1)$ proves (2). Finally, $l + 2 \le 3l$ for $l \ge 1$, so taking square roots and using $u = \varepsilon_{\text{mch}}/2$ gives Theorem 1 of the main text. □

The unrolled sum explains the role of moderate amplification: earlier errors traverse more layers, while the local mean-square error grows with N_j. The closed-form bound shows how their combined contribution depends on both effects; its constant increases as $\bar{c}$ approaches two, and Γ accounts for the transient layers. The profile bound (1) retains the second moments themselves; the closed-form bound replaces each ratio N_j/N_l by the bound $(C_N/c_N)(j+1)/(l+1)$ from Lemma 4 and the factors below l_0 by Γ. This preserves the rate, but the first replacement enlarges the term of index j by the factor $(C_N/c_N)\,(N_l/(l+1))/(N_j/(j+1)) \ge 1$, most for the early terms when the early layers grow more slowly than the later ones (Appendix E.8). The theorem bounds the mean-square absolute error by a quantity of order u^2l^2 and its ratio to N_l by a quantity of order u^2l. When local mean-square errors have comparable coefficients and cross terms are negligible, the same accumulation model predicts the square-root growth used to interpret the activation curves.

Comparing two floating-point executions. The curves in §5.2 compare two computed states, not a computed state with the real-arithmetic one. Let $\hat{f}_l^A$ and $\hat{f}_l^B$ be two floating-point implementations of the same layer, with local errors $\xi_l^r = \hat{f}_l^r(\hat{Y}_{l-1}^r) - f_l(\hat{Y}_{l-1}^r)$ for $r \in \{A, B\}$. Their difference decomposes as

$$D_l^{AB} = \hat{Y}_l^A - \hat{Y}_l^B = d_l + \rho_l^{AB}, \qquad d_l = \xi_l^A - \xi_l^B, \qquad \rho_l^{AB} = f_l(\hat{Y}_{l-1}^A) - f_l(\hat{Y}_{l-1}^B).$$

Suppose the input pairs satisfy Assumption 2, and Assumption 3 holds for (d_l, ρ_l^{AB}) in place of (ξ_l, ρ_l), with constants η_{AB} and c_{AB}. The argument of Theorem 7, now started from a possibly nonzero initial difference, gives

$$\mathbb{E}\|D_l^{AB}\|^2 \le \Psi_{0l}\,\mathbb{E}\|D_0^{AB}\|^2 + (1 + 2c_{AB})\,\eta_{AB}^2 u^2 \sum_{j=1}^{l} N_j \Psi_{jl}, \qquad \Psi_{jl} = \prod_{k=j+1}^{l} \Big(1 + \frac{c_k'}{k}\Big). \tag{3}$$

The initial term vanishes for a candidate that receives the reference input and equals the energy of the injected perturbation for the perturbed reference. For the latter, Ψ_{0l} includes the potentially large factor $1 + c_1'$ of the first layer. The constants describe the difference of the two local errors. Rounding that both executions perform on nearly equal operands largely cancels in d_l, so η_{AB} can be much smaller than the constant η that compares either execution with real arithmetic; Appendix E.7 quantifies this cancellation for FP8 quantization.

E.4 Activation-gradient errors

Activation gradients are derivatives of the loss with respect to intermediate activations, and their numerical errors depend on the values saved during the forward pass. We call this collection of saved values the forward trace. For each realization, let $\bar{\delta}_l$ denote exact evaluation of the backpropagation formulas and let $\hat{\delta}_l$ denote their floating-point evaluation on the same trace. This comparison isolates rounding errors in backpropagation; the trace itself remains random when we take expectations.

With the stored values fixed, each exact backward step is linear in its incoming gradient. Denote that map by $\mathcal{B}_k$ and define

$$\bar{\delta}_{k-1} = \mathcal{B}_k \bar{\delta}_k, \qquad \zeta_l = \hat{\delta}_l - \bar{\delta}_l, \qquad r_k = \hat{\delta}_{k-1} - \mathcal{B}_k \hat{\delta}_k.$$

Subtracting the exact and computed steps gives the error recurrence

$$\zeta_{k-1} = \mathcal{B}_k \zeta_k + r_k.$$

We normalize by the gradient's second moment $S_l = \mathbb{E}\|\bar{\delta}_l\|^2 > 0$, accounting for amplification of the gradient along with that of the error. The output gradient used to initialize this part of backpropagation is its *backward seed*; its rounding error is included separately below.

Assumption 4 (Rounding and error propagation in backpropagation)**.** *The seed and local errors satisfy*

$$\mathbb{E}\|\zeta_L\|^2 \le \sigma_0^2 u^2 S_L, \qquad \mathbb{E}\|r_k\|^2 \le \sigma_r^2 u^2 S_{k-1}, \qquad \mathbb{E}\langle \mathcal{B}_k \zeta_k, r_k\rangle \le c_B \mathbb{E}\|r_k\|^2.$$

Relative error propagation obeys

$$\frac{\mathbb{E}\|\mathcal{B}_k \zeta_k\|^2}{S_{k-1}} \le \mu_k \frac{\mathbb{E}\|\zeta_k\|^2}{S_k}, \qquad \prod_{k=l+1}^{m} \mu_k \le K_B \quad (0 \le l \le m \le L),$$

with $\mu_k, c_B, \sigma_0, \sigma_r \ge 0$ and $K_B \ge 1$. All bounding constants are uniform in depth.

The local moment bounds have the same role as in the forward analysis. The propagation condition compares amplification of the mean-square error with that of the gradient's second moment, allowing the two to differ as long as their accumulated ratio remains bounded. It is imposed on every interval because a rounding error introduced partway through the backward pass traverses only the remaining part of that path.

Theorem 8 (Rounding error in backpropagation)**.** *Under Assumption 4,*

$$\frac{\mathbb{E}\|\zeta_l\|^2}{S_l} \le K_B u^2 \big[\sigma_0^2 + (1 + 2c_B)(L - l)\sigma_r^2\big].$$

Consequently, for $\sigma = \max(\sigma_0, \sigma_r)$, $\mathcal{E}(\hat{\delta}_l, \bar{\delta}_l) = O(\sigma u \sqrt{L - l + 1})$.

Proof. Expanding the squared error and using the cross-moment bound gives

$$\mathbb{E}\|\zeta_{k-1}\|^2 \le \mathbb{E}\|\mathcal{B}_k\zeta_k\|^2 + (1+2c_B)\mathbb{E}\|r_k\|^2.$$

Divide by S_{k-1} and apply the propagation and local-error conditions to obtain

$$R_{k-1} \le \mu_k R_k + (1+2c_B)\sigma_r^2 u^2, \qquad R_k = \mathbb{E}\|\zeta_k\|^2/S_k.$$

Unrolling this scalar recurrence from the backward seed yields

$$R_l \le \left(\prod_{k=l+1}^{L}\mu_k\right)R_L + (1+2c_B)\sigma_r^2 u^2 \sum_{j=l+1}^{L}\prod_{k=l+1}^{j-1}\mu_k.$$

The first product propagates the seed error, while the term indexed by j propagates the error introduced in backward step j. Every product is at most K_B, including empty products, and the sum contains $L-l$ terms. Combining this with $R_L \le \sigma_0^2 u^2$ proves the stated inequality. Taking square roots and using $u = \varepsilon_{\mathrm{mch}}/2$ gives Theorem 2 of the main text. □

To recover the error of a complete computation, let δ_l be the gradient evaluated on the ideal forward trace and set $\theta_l = \bar{\delta}_l - \delta_l$. The decomposition $\hat{\delta}_l - \delta_l = \zeta_l + \theta_l$ and the triangle inequality in L^2 give

$$\mathcal{E}(\hat{\delta}_l, \delta_l) \le \sqrt{\frac{S_l}{\mathbb{E}\|\delta_l\|^2}}\,\mathcal{E}(\hat{\delta}_l, \bar{\delta}_l) + \frac{\sqrt{\mathbb{E}\|\theta_l\|^2}}{\sqrt{\mathbb{E}\|\delta_l\|^2}}. \tag{4}$$

The factor multiplying the first term converts its normalization from the RMS gradient norm on the stored trace to that on the ideal trace. The second term contains the effect of changed backward Jacobians and of the loss gradient recomputed from changed forward activations, so controlling it requires sensitivity of the backpropagation formulas to their saved values. We retain this contribution in the parameter-gradient analysis below and include both sources when analyzing the proxy, which perturbs the forward computation as well as the backward seed.

E.5 Parameter-gradient errors

Parameter gradients combine the forward and backward computations through sums of outer products. For a linear block acting on row vectors, let $A_l \in \mathbb{R}^{n\times r}$ contain its inputs and $B_l \in \mathbb{R}^{n\times s}$ its output gradients, placing the common loss-average factor outside B_l. With $G_l = G^{W^l}$,

$$G_l = \frac{1}{n}A_l^\top B_l.$$

Thus the activation factor is a normalized residual state for the first FFN matrix and the post-activation hidden vector for the second. A bias gradient is the case in which A_l is a column of ones, and the gradient of a normalization gain is the diagonal of $\frac{1}{n}A_l^\top B_l$ with A_l the normalized input; Theorem 9 below applies to these cases with each product replaced by its diagonal. To distinguish errors in these factors from errors in their multiplication, define

$$\Delta A_l = \hat{A}_l - A_l, \qquad \Delta B_l = \hat{B}_l - B_l, \qquad \Omega_l = \hat{G}_l - \frac{1}{n}\hat{A}_l^\top \hat{B}_l.$$

Expanding the product gives the exact decomposition

$$\hat{G}_l - G_l = \frac{1}{n}\left(\Delta A_l^\top B_l + A_l^\top \Delta B_l + \Delta A_l^\top \Delta B_l\right) + \Omega_l. \tag{5}$$

The three product terms describe the effects of perturbed factors, including their interaction, while Ω_l includes rounding in gradient formation and reduction.

An absolute bound on these terms does not by itself give a useful relative error if the outer products nearly cancel. We therefore control the size of their sum relative to the factor norms, together with the norm fluctuations needed when averaging products of errors and signals.

Assumption 5 (Gradient conditioning and fourth-moment bounds)**.** *For constants $K > 0$ and $M \ge 1$ independent of depth,*

$$\mathbb{E}\|A_l\|_F^2\,\mathbb{E}\|B_l\|_F^2 \le K^2\mathbb{E}\|A_l^\top B_l\|_F^2, \qquad \mathbb{E}\|T\|_F^4 \le M\left(\mathbb{E}\|T\|_F^2\right)^2$$

for $T \in \{A_l, B_l, \Delta A_l, \Delta B_l\}$.

The constant K is a conditioning factor for the bilinear product at the RMS scale: a small gradient formed from large factors requires a large K and admits a correspondingly larger relative error. Its definition accounts for cancellation between outer products. The normalized fourth-moment bound controls fluctuations of the squared tensor norms when averaging products of errors and signals, without imposing independence between factors or between sequence positions.

Theorem 9 (Parameter-gradient error)**.** *Define the nonnegative relative error quantities by*

$$\alpha_A^2(l) = \frac{\mathbb{E}\|\Delta A_l\|_F^2}{\mathbb{E}\|A_l\|_F^2}, \qquad \alpha_B^2(l) = \frac{\mathbb{E}\|\Delta B_l\|_F^2}{\mathbb{E}\|B_l\|_F^2}, \qquad \omega_l = \frac{\sqrt{\mathbb{E}\|\Omega_l\|_F^2}}{\sqrt{\mathbb{E}\|G_l\|_F^2}},$$

and the coherence ratios

$$r_A(l) = \frac{\mathbb{E}\|\Delta A_l^\top B_l\|_F^2 \, \mathbb{E}\|A_l\|_F^2}{\mathbb{E}\|\Delta A_l\|_F^2 \, \mathbb{E}\|A_l^\top B_l\|_F^2}, \qquad r_B(l) = \frac{\mathbb{E}\|A_l^\top \Delta B_l\|_F^2 \, \mathbb{E}\|B_l\|_F^2}{\mathbb{E}\|\Delta B_l\|_F^2 \, \mathbb{E}\|A_l^\top B_l\|_F^2},$$

$$r_{AB}(l) = \frac{\mathbb{E}\|\Delta A_l^\top \Delta B_l\|_F^2 \, \mathbb{E}\|A_l\|_F^2 \, \mathbb{E}\|B_l\|_F^2}{\mathbb{E}\|\Delta A_l\|_F^2 \, \mathbb{E}\|\Delta B_l\|_F^2 \, \mathbb{E}\|A_l^\top B_l\|_F^2},$$

where r_A is set to zero when $\Delta A_l = 0$, r_B when $\Delta B_l = 0$, and r_{AB} when either vanishes. If $\mathbb{E}\|G_l\|_F^2 > 0$, then

$$\mathcal{E}(\hat{G}_l, G_l) \le \sqrt{r_A(l)}\,\alpha_A(l) + \sqrt{r_B(l)}\,\alpha_B(l) + \sqrt{r_{AB}(l)}\,\alpha_A(l)\alpha_B(l) + \omega_l. \tag{6}$$

Under Assumption 5, each coherence ratio is at most $K^2 M$, so

$$\mathcal{E}(\hat{G}_l, G_l) \le K\sqrt{M}\left[\alpha_A(l) + \alpha_B(l) + \alpha_A(l)\alpha_B(l)\right] + \omega_l.$$

Proof. Since $n^2\mathbb{E}\|G_l\|_F^2 = \mathbb{E}\|A_l^\top B_l\|_F^2$, the definitions give

$$\frac{\mathbb{E}\|\Delta A_l^\top B_l\|_F^2}{n^2\mathbb{E}\|G_l\|_F^2} = r_A(l)\,\alpha_A^2(l), \qquad \frac{\mathbb{E}\|A_l^\top \Delta B_l\|_F^2}{n^2\mathbb{E}\|G_l\|_F^2} = r_B(l)\,\alpha_B^2(l),$$

$$\frac{\mathbb{E}\|\Delta A_l^\top \Delta B_l\|_F^2}{n^2\mathbb{E}\|G_l\|_F^2} = r_{AB}(l)\,\alpha_A^2(l)\alpha_B^2(l).$$

Taking square roots and applying the triangle inequality in L^2 to (5) yields (6), with ω_l accounting for the formation error. This step allows arbitrary correlations among the four contributions.

For the second bound, let U, V be any pair of factors among those appearing in (5). Submultiplicativity of the Frobenius norm and Cauchy–Schwarz give

$$\begin{aligned}\mathbb{E}\|U^\top V\|_F^2 &\le \mathbb{E}[\|U\|_F^2\|V\|_F^2] \\ &\le \sqrt{\mathbb{E}\|U\|_F^4\,\mathbb{E}\|V\|_F^4} \le M\,\mathbb{E}\|U\|_F^2\,\mathbb{E}\|V\|_F^2.\end{aligned}$$

With $(U, V) = (\Delta A_l, B_l)$ and the conditioning inequality,

$$r_A(l) \le M\,\frac{\mathbb{E}\|A_l\|_F^2\,\mathbb{E}\|B_l\|_F^2}{\mathbb{E}\|A_l^\top B_l\|_F^2} \le MK^2,$$

and the pairs $(A_l, \Delta B_l)$ and $(\Delta A_l, \Delta B_l)$ give $r_B(l) \le MK^2$ and $r_{AB}(l) \le MK^2$ in the same way. □

The coherence ratios compare how coherently the error products add over the tokens of a batch with how coherently the gradient's own products do; they also reflect how the errors align with the factors, and they are unrelated to the mutual coherence of compressed sensing. With $\Theta(U, V) = n\,\mathbb{E}\|U^\top V\|_F^2/(\mathbb{E}\|U\|_F^2\,\mathbb{E}\|V\|_F^2)$, which lies between zero and nM under the fourth-moment bound of Assumption 5, one has $r_A = \Theta(\Delta A_l, B_l)/\Theta(A_l, B_l)$, and similarly for the other ratios. The bound K^2M is the worst case, in which the error products add up as coherently as the fourth-moment bound permits while the gradient cancels as much as the conditioning bound permits. Factor errors that are less coherent across tokens than the gradient give ratios below one and are partly averaged out in the

token sum; shared components of the factor errors, as discussed in Appendix E.6, raise the ratios. Like K and M, the ratios can be estimated from known-correct runs (Appendix E.8).

To obtain the depth dependence in the main text, we must relate the factor errors to the residual-state and backward errors already bounded. Let $\bar{B}_l$ denote exact formation of the gradient factor on the stored forward trace, using the exact backward values on that trace, and define its forward-trace contribution by

$$t_l = \frac{\sqrt{\mathbb{E}\|\bar{B}_l - B_l\|_F^2}}{\sqrt{\mathbb{E}\|B_l\|_F^2}}.$$

We state the required perturbation bounds for the block computations in RMS relative error.

Assumption 6 (Factor-error stability)**.** *For constants $L_A, L_B \geq 0$ independent of depth,*

$$\begin{aligned} \alpha_A(l) &\leq L_A\big[\mathcal{E}(\hat{Y}_{l-1}, Y_{l-1}) + u\big], \\ \alpha_B(l) &\leq L_B\big[\mathcal{E}(\hat{\delta}_l, \bar{\delta}_l) + u\big] + t_l, \end{aligned} \tag{7}$$

and the forward-trace term and the formation error satisfy $t_l = O(u\sqrt{L})$ and $\omega_l = O(u)$ uniformly in l.

For the second inequality, the decomposition $\Delta B_l = (\hat{B}_l - \bar{B}_l) + (\bar{B}_l - B_l)$ separates arithmetic and incoming-gradient errors on a fixed trace from the term t_l; the condition on L_B concerns the former contribution, including the conversion to the ideal factor's RMS scale. The forward-trace condition follows from Theorem 7 when the map from the saved forward values to the gradient factor has a relative Lipschitz constant independent of depth.

For the activation factor, the normalization calculation supplies a concrete sufficient condition. Consider a projection whose input is $A_l = \mathrm{LayerNorm}(Y_{l-1})$ with $l \geq 2$, and let $q_l = \hat{A}_l - \mathrm{LayerNorm}(\hat{Y}_{l-1})$ be its local normalization error. Suppose that the comparison segments satisfy the denominator condition of Lemma 5, with $\bar{s}^2 = N_{l-1}/(nd)$, and that this LayerNorm satisfies a signal condition of the form in Lemma 4. The finite-difference bound and the triangle inequality give

$$\alpha_A(l) \leq \frac{1 + d^{-1/2}}{\sqrt{\nu}\,\bar{s}} \sqrt{\frac{N_{l-1}}{\mathbb{E}\|A_l\|_F^2}}\, \mathcal{E}(\hat{Y}_{l-1}, Y_{l-1}) + \frac{\sqrt{\mathbb{E}\|q_l\|_F^2}}{\sqrt{\mathbb{E}\|A_l\|_F^2}}.$$

Here $N_{l-1}/\bar{s}^2 = nd$, while $\mathbb{E}\|A_l\|_F^2 \geq \kappa nd$ by Lemma 5 and the signal condition. If the local error has RMS relative magnitude at most $c_q u$, it follows that

$$\alpha_A(l) \leq \frac{1 + d^{-1/2}}{\sqrt{\nu\kappa}}\, \mathcal{E}(\hat{Y}_{l-1}, Y_{l-1}) + c_q u,$$

with a coefficient at most $2/\sqrt{\nu\kappa}$, which is uniform in l and d when ν and κ are bounded below. In the first layer the input $\hat{Y}_0 = Y_0$ is exact, so the factor $\mathrm{LayerNorm}(Y_0)$ has $\alpha_A(1) \leq c_q u$. For the first FFN matrix, whose activation factor is $\mathrm{LayerNorm}(Z_l)$, the same argument with $\bar{s}^2 = \mathbb{E}\|Z_l\|_F^2/(nd)$ and the signal condition of the FFN LayerNorm gives the bound with $\mathcal{E}(\hat{Z}_l, Z_l)$ in place of $\mathcal{E}(\hat{Y}_{l-1}, Y_{l-1})$. Write $\xi_l^{\mathcal{A}} = \hat{Z}_l - [\hat{Y}_{l-1} + \mathcal{A}_l(\hat{Y}_{l-1})]$ for the local error of the attention sublayer and $N_l^Z = \mathbb{E}\|Z_l\|_F^2 \geq N_{l-1}$. The first step of the proof of Lemma 6 gives

$$\mathcal{E}(\hat{Z}_l, Z_l) \leq \sqrt{1 + a_l/l}\; \mathcal{E}(\hat{Y}_{l-1}, Y_{l-1}) + \frac{\sqrt{\mathbb{E}\|\xi_l^{\mathcal{A}}\|_F^2}}{\sqrt{N_l^Z}},$$

so only the local attention error is an additional assumption. This verifies the first relation in (7) for the normalization factors; for the other block operations, it is part of Assumption 6.

Applying Theorems 7 and 8 to (7) gives

$$\alpha_A(l) = O(u\sqrt{l}), \qquad \alpha_B(l) = O(u\sqrt{L - l + 1}) + t_l.$$

Under Assumption 6, the sum of the first-order factor errors is $O(u\sqrt{L})$, using $\sqrt{l} + \sqrt{L-l+1} \le \sqrt{2(L+1)}$. Their product is $O(u^2 L)$, so Theorem 9, with coherence ratios bounded uniformly in depth as under Assumption 5, gives

$$\mathcal{E}(\hat{G}_l, G_l) \le C_1 u\sqrt{L} + C_2 u^2 L.$$

When $u\sqrt{L}$ is small, the second term is of higher order, yielding the uniform upper envelope in Theorem 3. The envelope combines accumulation in both directions while allowing individual layerwise errors to vary beneath it. For a shared parameter, all block contributions must be combined before assessing conditioning, since cancellation between uses can also affect its gradient norm.

Formation error from rounding cells. The formation error ω_l can be estimated directly from a tensor's rounding cells. For a gradient product $C = A^\top B/n$ whose formation error is dominated by its final storage cast, model each entry's rounding error as uniform on an interval centered at zero whose width h_{ij} is the local spacing of the output format. Its second moment is then $h_{ij}^2/12$, and summing over entries gives

$$\omega_{\text{cast}}^2 \approx \frac{\sum_{ij} h_{ij}^2}{12\|C\|_F^2}. \tag{8}$$

This estimate uses only marginal second moments and does not require independent matrix entries. In the 24-layer BF16 GPT traced for Table 2, the W2 formation error of each microbatch, measured against an FP64 product of the stored BF16 factors, is 0.166%, or $0.42u_{\text{BF16}}$, in every layer, and (8) agrees with it to within 0.24% (median 0.07%). The weight-gradient kernel rounds each product to BF16 before it is accumulated in FP32 over microbatches, so the formation error has the BF16 scale although the accumulated gradient is stored in FP32. For a fused normalization–linear operation, A is the normalized input to the matrix multiplication, not the module input before normalization, and the gradient identity includes the loss-average and accumulation factors.

E.6 First-order analysis of the perturbation proxy

The preceding bounds describe how numerical errors accumulate, but their constants depend on the tensor and its computation. The perturbation proxy addresses this dependence by measuring a response of the reference run itself. For a traced tensor T, write

$$e_T = \|T^H - T^F\|, \qquad \tau_T = \|T^{F,+} - T^F\|.$$

Here H is the candidate, F is the reference, and $F,+$ is the perturbed reference. The algorithm divides both differences by the same reference norm, so this normalization cancels in their ratio; a tensor with $\tau_T = 0$ is flagged whenever $e_T > 0$. In the backward comparison, the added seed perturbation is applied to the loss gradient recomputed on the perturbed forward trace. In the implementation this is the gradient with respect to the logits; it reaches the final residual state through the transposed Jacobian of the output layer, which is part of the map $\mathcal{T}$ below.

Two runs are aligned when they execute the same real-arithmetic computation with the same discrete choices, such as routing and masking. For aligned runs, represent each local arithmetic difference by an additive error at its operation boundary, the output of the floating-point kernel or storage cast that introduces it. The resulting map for the traced tensor is $\mathcal{T}(z)$, where $z = (X, g, \xi_1, \dots, \xi_m)$ collects the input, the added backward-seed perturbation g, and the local error variables. The loss gradient is computed inside this map from the forward trace; g is the additional perturbation applied to it. The ideal point is $z_0 = (X, 0, 0, \dots, 0)$.

We apply Taylor's theorem to this augmented real-arithmetic map, with rounding decisions entering through the values of the ξ_k. When an error is reused, it appears as one variable whose derivative includes every path by which that error reaches T. This representation is particularly useful for gradients, where a forward error can change both an activation factor and the derivative propagated back through it.

Lemma 10 (First-order expansion of a traced tensor)**.** *Suppose $\mathcal{T}$ has an H_T-Lipschitz Jacobian on a convex neighborhood containing z_0, z_A, z_B. Then*

$$T^A - T^B = J_X \Delta X + J_g \Delta g + \sum_k \Pi_k (\xi_k^A - \xi_k^B) + R_T,$$

where $\Delta X = X^A - X^B$, $\Delta g = g^A - g^B$, *and* $J_X = D_X\mathcal{T}(z_0)$, $J_g = D_g\mathcal{T}(z_0)$, $\Pi_k = D_{\xi_k}\mathcal{T}(z_0)$. *The remainder satisfies*

$$\|R_T\| \le \frac{H_T}{2}\big(\|z_A - z_0\|^2 + \|z_B - z_0\|^2\big).$$

Proof. Put $h_A = z_A - z_0$. The fundamental theorem of calculus along the segment from z_0 to z_A gives

$$\mathcal{T}(z_A) = \mathcal{T}(z_0) + D\mathcal{T}(z_0)h_A + r_A, \qquad r_A = \int_0^1 \big[D\mathcal{T}(z_0 + th_A) - D\mathcal{T}(z_0)\big]h_A\,\mathrm{d}t.$$

The Lipschitz condition on the Jacobian therefore implies

$$\|r_A\| \le \int_0^1 H_T t\|h_A\|^2\,\mathrm{d}t = \frac{H_T}{2}\|z_A - z_0\|^2.$$

The same expansion at z_B has remainder bounded by $H_T\|z_B - z_0\|^2/2$. Subtracting the expansions gives the linear term $D\mathcal{T}(z_0)(z_A - z_B)$, whose coordinate blocks are precisely the stated input, seed, and local-error contributions. With $R_T = r_A - r_B$, the triangle inequality gives the remainder bound. □

The operators are the Jacobian blocks of the augmented computation, which the chain rule identifies in familiar cases. For an error added after forward layer k and observed at Y_l, the operator is $\Pi_k = J_l \cdots J_{k+1}$, with $J_m = Df_m(Y_{m-1})$ the layer Jacobians on the ideal trace. For a local error reaching a parameter gradient $G = A^\top B/n$ through its factors, its derivative in a direction v is

$$\Pi_k^G v = \frac{1}{n}\left[(D_{\xi_k}A[v])^\top B + A^\top (D_{\xi_k}B[v])\right].$$

Thus the two effects of a shared forward error are combined in one operator before taking a second moment. The same construction includes changes in saved activations, backward Jacobians, and the recomputed loss gradient.

For fixed depth, perturbations with $\|z_A - z_0\|, \|z_B - z_0\| = O(u)$ have a remainder of order u^2 under the lemma's hypotheses. Its constant can depend on depth, so the comparison below is used when the first-order approximation is accurate at the scale of the measured response. Within that regime, a common Jacobian transfers a local mean-square comparison to the traced tensor if the proxy's perturbations cover the relevant directions. All tensors in the following second-moment matrices are vectorized.

Assumption 7 (Second-moment comparison for local errors)**.** *In the probabilistic model conditional on the ideal trace, let* $d_k^e = \xi_k^H - \xi_k^F$ *and* $d_k^\tau = \xi_k^{F,+} - \xi_k^F$. *For distinct error groups* $j \ne k$, *assume*

$$\mathbb{E}[d_j^e(d_k^e)^\top] = 0, \qquad \mathbb{E}[d_j^\tau(d_k^\tau)^\top] = 0.$$

The external perturbations satisfy $\mathbb{E}[\Delta X(d_k^\tau)^\top] = 0$ *and* $\mathbb{E}[\Delta g(d_k^\tau)^\top] = 0$ *for every* k. *For a common constant* $C_{\mathrm{loc}} > 0$,

$$\Sigma_k^e := \mathbb{E}[d_k^e(d_k^e)^\top] \preceq C_{\mathrm{loc}}\mathbb{E}[d_k^\tau(d_k^\tau)^\top] =: C_{\mathrm{loc}}\Sigma_k^\tau.$$

The groups include forward and backward arithmetic and gradient formation; repeated uses of a local error belong to the same group.

The matrices are uncentered second-moment matrices, and $\preceq$ denotes the positive semidefinite order. They include any bias in the local differences and coincide with covariance matrices when those differences have mean zero. In scalar form, the comparison requires $\mathbb{E}[\langle v, d_k^e\rangle^2] \le C_{\mathrm{loc}}\mathbb{E}[\langle v, d_k^\tau\rangle^2]$ for every direction v; comparable total norms alone would not ensure this after propagation by an arbitrary operator. The two comparison families may be dependent through their shared reference run, because the argument only requires the stated cross moments within each family.

Theorem 11 (Mean-square comparison of the linearized differences)**.** *Let* $e_T^{(1)}$ *and* $\tau_T^{(1)}$ *be the norms of the respective linearized differences in Lemma 10. Under Assumption 7,*

$$\mathbb{E}[(e_T^{(1)})^2] \le C_{\mathrm{loc}}\mathbb{E}[(\tau_T^{(1)})^2]. \tag{9}$$

This holds for activations, activation gradients, and parameter gradients. If the same constant holds uniformly over the ideal traces, it also holds after averaging over those traces.

Proof. Conditional on the ideal trace, the propagation operators are fixed. The candidate and reference have the same external input and no added seed perturbation, so their linear difference is $\sum_k \Pi_k d_k^e$. For any family of injection vectors,

$$\mathbb{E}\left\|\sum_k \Pi_k d_k\right\|^2 = \sum_{j,k} \operatorname{tr}\left(\Pi_j \mathbb{E}[d_j d_k^\top]\Pi_k^\top\right).$$

Under Assumption 7, all terms with $j \neq k$ vanish. The proxy additionally contains $J_X\Delta X + J_g\Delta g$, whose cross terms with the error groups also vanish. Consequently,

$$\mathbb{E}[(e_T^{(1)})^2] = \sum_k \operatorname{tr}(\Pi_k \Sigma_k^e \Pi_k^\top), \qquad \mathbb{E}[(\tau_T^{(1)})^2] = I_T + \sum_k \operatorname{tr}(\Pi_k \Sigma_k^\tau \Pi_k^\top),$$

where $I_T = \mathbb{E}\|J_X\Delta X + J_g\Delta g\|^2 \geq 0$. This term retains any dependence between the input and seed perturbations.

To compare the sums, put $D_k = C_{\text{loc}}\Sigma_k^\tau - \Sigma_k^e \succeq 0$. For every vector v,

$$v^\top \Pi_k D_k \Pi_k^\top v = (\Pi_k^\top v)^\top D_k (\Pi_k^\top v) \geq 0.$$

Thus $\Pi_k D_k \Pi_k^\top$ is positive semidefinite and has nonnegative trace, which gives $\operatorname{tr}(\Pi_k \Sigma_k^e \Pi_k^\top) \leq C_{\text{loc}} \operatorname{tr}(\Pi_k \Sigma_k^\tau \Pi_k^\top)$. Summing and using $I_T \geq 0$ yields

$$\mathbb{E}[(e_T^{(1)})^2] \leq C_{\text{loc}} \sum_k \operatorname{tr}(\Pi_k \Sigma_k^\tau \Pi_k^\top) \leq C_{\text{loc}} \mathbb{E}[(\tau_T^{(1)})^2].$$

If C_{loc} is uniform over ideal traces, averaging this conditional inequality over those traces proves the unconditional statement. □

The comparison leaves the propagation operators arbitrary, but requires the proxy to cover the candidate's local error directions. A direction with zero proxy second moment and positive candidate second moment would violate the assumption, regardless of how similar their total error norms were. As a simple sufficient model, two independent, zero-mean rounding realizations with covariance Σ have difference covariance 2Σ; if both comparison pairs have this local behavior, $C_{\text{loc}} = 1$ suffices. Correlated roundings or different implementations require their own comparison constant.

To relate the first-order comparison to measured differences, let $r_e = \sqrt{\mathbb{E}\|R_T^e\|^2}$ and $r_\tau = \sqrt{\mathbb{E}\|R_T^\tau\|^2}$ be the RMS remainders in the two comparisons. Applying the triangle inequality before and after Theorem 11 gives

$$\begin{aligned}
\sqrt{\mathbb{E}e_T^2} &\leq \sqrt{\mathbb{E}[(e_T^{(1)})^2]} + r_e \\
&\leq \sqrt{C_{\text{loc}}}\sqrt{\mathbb{E}[(\tau_T^{(1)})^2]} + r_e \\
&\leq \sqrt{C_{\text{loc}}}\sqrt{\mathbb{E}\tau_T^2} + r_e + \sqrt{C_{\text{loc}}}\, r_\tau.
\end{aligned} \tag{10}$$

This makes the approximation explicit: a multiplicative comparison is informative when the remainder terms are small at the scale of the proxy response. Even then, a comparison of second moments does not bound every realized ratio e_T/τ_T. The theorem therefore motivates the tolerance rule, while the safety factor is selected from measured correct-run ratios as described in Appendix C.2.

Lemma 10 holds with any point of the neighborhood as the base point in place of z_0. Expanding both comparisons around the point z_F of the reference execution evaluates the Jacobian blocks on the reference trace and bounds the remainders by $\frac{H_T}{2}\|z_H - z_F\|^2$ and $\frac{H_T}{2}\|z_{F,+} - z_F\|^2$, which involve only the separations of the paired executions. If Assumption 7 holds conditional on the reference execution, with a uniform comparison constant, Theorem 11 applies with these reference-based Jacobian blocks. This is an additional hypothesis: conditioning on the reference rounding errors can create cross moments that vanish conditional on the ideal trace alone. This form is preferable when each execution deviates from real arithmetic by much more than the executions differ from each other, as for FP8 quantization on a common grid.

Rounding errors correlated across tokens. A parameter gradient is a sum over the tokens of a batch, so its difference between two executions depends on how the contributions of different tokens are correlated, not only on their sizes. In the notation of Appendix E.5, with the layer index dropped, $G = \frac{1}{n}\sum_i A_{i,:}^\top B_{i,:}$, and the first-order difference of G is $\frac{1}{n}\sum_i E_i$ with $E_i = A_{i,:}^\top \Delta B_{i,:} + \Delta A_{i,:}^\top B_{i,:}$. If the summands have mean zero and are uncorrelated across tokens, $\mathbb{E}\|\frac{1}{n}\sum_i E_i\|_F^2 = \frac{1}{n^2}\sum_i \mathbb{E}\|E_i\|_F^2$, which decreases as $1/n$ when their energies are comparable. A component common to many summands, such as the product of the token means of A and ΔB, does not decrease with n. Bias gradients, of the linear maps and of LayerNorm, are token means of gradient rows, so a common component of the row differences enters them in full.

Algorithm 1 draws its perturbations isotropically over all entries of a tensor, but shared computation, such as attention or a reduction over tokens, can correlate their responses. Whether the perturbed reference reproduces a common component of the candidate's difference is therefore a question about directional second moments, as in Assumption 7. One mechanism for such a component is rounding of operands that are nearly equal across tokens. In the output layer, with vocabulary size V, output-embedding rows w_c, predicted probabilities p_{ic}, and targets y_i, the backward product has rows

$$b_i = \sum_{c=1}^{V} p_{ic} w_c - w_{y_i},$$

up to the loss-average factor. When the output distribution is nearly uniform, $p_{ic} \approx 1/V$, the first term is nearly the same vector for all tokens and small compared with the target term. The target term is exactly representable in the output format when the embedding rows are stored in that format and the loss-average factor is a power of two. Where the first term is smaller than half the spacing of the floating-point numbers near the target term, rounding removes it, and the partitioning of the sum over the vocabulary, for example across tensor-parallel ranks, determines where this happens. Logits of order one make the first term vary between tokens, which reduces the common component without removing it.

E.7 FP8 quantization with shared scaling factors

FP8 recipes quantize the operands of matrix multiplications to an eight-bit format, such as E4M3 with three explicit significand bits, and form the products from the quantized operands. For a scaling factor $\sigma > 0$, write

$$Q_\sigma(x) = \sigma^{-1}\,\mathrm{rd}(\sigma x),$$

where rd rounds each entry to the nearest value representable in the format and saturates at its largest finite value. Delayed scaling sets σ from the largest magnitude (amax) recorded in previous passes; current scaling computes it from the tensor being quantized. Two executions that compute the same tensor apply such a map to nearby operands, so a local difference d_k of Assumption 7 at a quantizer is a difference of two rounding errors. Its size depends on whether the two executions round on a common grid.

Within a binade, the values $\sigma^{-1}\,\mathrm{rd}(\sigma x)$ are equally spaced, and the rounding boundaries lie midway between consecutive values. For an entry x whose scaled value σx is normal and below the saturation threshold, the spacing h satisfies $\varepsilon_Q|x|/2 < h \le \varepsilon_Q|x|$, where ε_Q is the machine epsilon of the format: 2^{-3} for E4M3 and 2^{-2} for E5M2. The *phase* of x is its position relative to the rounding boundaries, in units of h.

Lemma 12 (Rounding nearby operands with a common grid)**.** *Let x and $x+\Delta$ be rounded by the same map Q_σ. Suppose that, on an interval containing $x+t$ for all $|t| \le h$, the representable values, and hence the rounding boundaries, are spaced h apart, and that $|\Delta| \le h$. If, conditional on Δ, the phase of x is uniformly distributed on $[0,1)$, then*

$$\mathbb{E}\big[Q_\sigma(x+\Delta) - Q_\sigma(x)\,\big|\,\Delta\big] = \Delta, \qquad \mathbb{E}\big[(Q_\sigma(x+\Delta) - Q_\sigma(x))^2\,\big|\,\Delta\big] = |\Delta|\,h.$$

Consequently, the difference of the two rounding errors, $\delta = \big(Q_\sigma(x+\Delta) - x - \Delta\big) - \big(Q_\sigma(x) - x\big)$, satisfies $\mathbb{E}[\delta \mid \Delta] = 0$ and $\mathbb{E}[\delta^2 \mid \Delta] = |\Delta|\,(h - |\Delta|)$.

Proof. Let b be the largest rounding boundary not exceeding x, and write $x = b + \varphi h$ with φ uniform on $[0,1)$. For $0 < \Delta \le h$, the two operands round to different values exactly when $x + \Delta \ge b + h$,

that is, when $\varphi \geq 1 - \Delta/h$. This event has probability Δ/h, and the two rounded values then differ by one spacing h. Hence $Q_\sigma(x+\Delta) - Q_\sigma(x)$ equals h with probability Δ/h and zero otherwise, which gives both moments. The case $\Delta < 0$ is symmetric, and $\Delta = 0$ is trivial. The statements for δ follow from $\delta = Q_\sigma(x+\Delta) - Q_\sigma(x) - \Delta$. □

The lemma refines the model of two independent roundings discussed after Theorem 11: when the operands are close and the phase hypothesis holds, the paired difference is unbiased, and its second moment is proportional to the separation $|\Delta|$ rather than to h^2. Applying it to each entry of a tensor whose entries satisfy its hypotheses, and using $h_i \leq \varepsilon_Q |x_i|$ and the Cauchy–Schwarz inequality for the sum and for the expectation,

$$\frac{\mathbb{E}\|Q_\sigma(x+\Delta) - Q_\sigma(x)\|^2}{\mathbb{E}\|x\|^2} = \frac{\mathbb{E}\sum_i |\Delta_i|\, h_i}{\mathbb{E}\|x\|^2} \leq \varepsilon_Q\, \varrho, \qquad \varrho = \left(\frac{\mathbb{E}\|\Delta\|^2}{\mathbb{E}\|x\|^2}\right)^{1/2}. \tag{11}$$

Quantization therefore enlarges a small relative separation ϱ of the operands to at most $\sqrt{\varepsilon_Q \varrho}$, a factor of $\sqrt{\varepsilon_Q/\varrho}$, and not to the scale ε_Q of the quantization error itself. For E4M3 operands with $\varrho = 2^{-8}$, half a BF16 machine epsilon, the factor is at most $\sqrt{32} \approx 5.7$. In a simulation with normally distributed entries x_i, the factor is 3.8 when Δ is independent of x and normally distributed, and 4.8 when $\Delta = \varrho x$. The phase hypothesis excludes separations that are themselves rounding errors on a grid containing the FP8 rounding boundaries: when $x + \Delta$ is the BF16 rounding of x, the separation is determined by the position of x on the BF16 grid, and the simulated factor is 9.8, above the bound 8.7 for the same ϱ. With this qualification, the lemma is consistent with the FP8 curves of Figure 4, in which executions that share scaling factors differ by a few BF16 units, whereas each of them differs from real arithmetic by a relative error of order $\varepsilon_Q/\sqrt{12}$ in every quantized operand.

Scaling factors. Lemma 12 requires a common grid. Two executions share the grid when their scaling factors are equal, and also when the factors differ by a power of two: for a binary format, $\mathrm{rd}(2^k y) = 2^k \mathrm{rd}(y)$ whenever y and $2^k y$ are normal and below the saturation threshold, so $Q_{2^k\sigma}(x) = Q_\sigma(x)$ for such entries. Otherwise, write the ratio of the factors as $1 + \epsilon_\sigma$. Since $Q_{(1+\epsilon_\sigma)\sigma}(y) = (1+\epsilon_\sigma)^{-1} Q_\sigma\big((1+\epsilon_\sigma)y\big)$, the difference of the rounding errors of $Q_{(1+\epsilon_\sigma)\sigma}(x+\Delta)$ and $Q_\sigma(x)$ equals the quantity δ of the lemma with Δ replaced by $\Delta + \epsilon_\sigma(x+\Delta)$, up to a term of relative size ϵ_σ. A mismatch of the factors thus acts as an additional relative separation ϵ_σ in every entry. Once $|\epsilon_\sigma|$ is comparable with ε_Q, the boundaries of the two grids are displaced by a substantial fraction of a cell, by amounts that vary between binades. If the two rounding errors are then modeled as independent and uniform on cells of widths h_H and h_F, their difference has second moment $(h_H^2 + h_F^2)/12$, approximately $h^2/6$ for comparable spacings, whatever the separation Δ; a mismatch of the factors makes such decorrelation plausible but does not by itself establish it.

These facts describe the marginal second moments of the local differences at FP8 quantizers. They do not determine the constant C_{loc} of Assumption 7, which compares second moments in every direction and also requires the cross-group conditions. Amax reduction over a parallel group gives all shards of a tensor the factor of the full tensor, but it does not make the factors of two runs equal: those differ by the relative difference of the runs' amaxes, which is of the order of the separation of their operands. The perturbed reference and a candidate that reduces amaxes over the full tensor therefore both satisfy the comparison with ϵ_σ of that order, and the marginal moments of their local differences follow Lemma 12, under its phase hypothesis, with separations of the order of the storage roundoff. Power-of-two factors, and block scaling with power-of-two block factors whose blocks do not cross shard boundaries, give common grids except where the amaxes of the runs fall on different sides of a power of two. A candidate that computes its scaling factors from shards of a tensor, as per-tensor current scaling does for the shards of row-parallel inputs and weights under tensor parallelism and for token shards under context parallelism, uses factors that differ from the reference's by the relative gap between the maximum of the shard and that of the full tensor. This gap need not be small compared with ε_Q. In the independent-rounding model, the candidate's local differences are then of order $h^2/6$ while the proxy's remain of order $|\Delta|h$, so at the affected quantizers even the marginal comparison requires a constant of order ε_Q/ϱ. How much this enlarges the difference of a traced tensor depends on the share of that difference contributed by the affected quantizers.

In controlled 16- and 32-layer GPTs built from TransformerEngine layers, with the FP8 recipe of the experiments (Appendix C.1), the perturbations of Algorithm 1, and tensor parallelism of size two, scaling factors computed per shard, as well as delayed scaling without amax reduction, raised

the median ratio e_T/τ_T of the block outputs to about 1.0, compared with 0.8–0.9 for the common grids of the other schemes. No traced tensor of these correct comparisons had a ratio above 1.4.

Initialization of delayed scaling. Delayed scaling starts from $\sigma = 1$ and sets each factor from the amax recorded in preceding passes. With $\sigma = 1$, E4M3 maps magnitudes below 2^{-10}, half its least subnormal value, to zero, and most gradient entries of the first backward pass are below this threshold. The first microbatches of an iteration are therefore quantized with factors that do not reflect the tensors (Appendix C.1). These effects are shared by the reference, the perturbed reference, and the candidate, but they make the first microbatches unrepresentative of FP8 training, and the FP8 curves show the last of the four microbatches of the traced iteration. In a controlled 32-layer GPT with the layer structure of Megatron-LM, repeating the comparison after a warm-up iteration, in which all factors are set from recorded maxima, changed the median ratio e_T/τ_T of the block outputs by at most 0.06 and the per-layer errors by at most 12%.

E.8 Validating the bounds

The constants of the bounds fall into two groups. Those of the growth lemmas follow from the initialization; the others describe the arithmetic and can be estimated from a few known-correct runs. This section evaluates both and checks the resulting bounds on held-out models. The tolerances of Algorithm 1 do not use these bounds; their safety margin is calibrated on the evaluated configurations in Appendix C.2.

Constants from the initialization. For FFN width $4d$, the proof of Lemma 4 bounds the attention branch by $(\sqrt{d}+1)^2/9$ and the FFN by $(d+1/3)/9$ per token, which together give Λ; the lower bound λ uses the FFN alone. The signal constant κ is not derived. In FP32 measurements of one 64-layer model for each $d \in \{512, 1024\}$, the token average of $\|z_i\|^2/d$ was at least 0.994 at every LayerNorm input that contains a branch output. With $\kappa = 0.994$ and $d = 1024$, $\lambda \approx 28.3$ and $\Lambda \approx 235$, so $C_N/c_N = 2\Lambda/\lambda \approx 16.6$ in (2). The closed-form bound evaluated with these values is therefore a conditional estimate; the profile bound does not use κ. The lower bound λ is fairly tight: the measured FFN outputs had squared norm at least about $0.033d$ per token, compared with $\lambda \approx 0.028d$.

Constants from calibration runs. With $\widehat{\mathbb{E}}$ denoting an average over calibration runs, the forward moments suggest

$$\hat{\eta}^2 = \max_l \frac{\widehat{\mathbb{E}}\|\xi_l\|^2}{u^2\widehat{\mathbb{E}}\|Y_l\|^2}, \qquad \hat{c}_F = \max_l \left(\frac{\widehat{\mathbb{E}}\langle \xi_l, \rho_l \rangle}{\widehat{\mathbb{E}}\|\xi_l\|^2} \right)_+ ,$$

over nonzero denominators, where $(x)_+ = \max(x, 0)$. Recomputing each layer in higher precision at its computed input approximates the real-arithmetic evaluation and gives ξ_l and ρ_l. Backward calibration evaluates the backpropagation formulas on the stored forward operands, including stored parameter values; recomputing the forward pass at higher precision would change the trace and mix the two error sources in (4). The mean-square amplification factors, the gradient conditioning, and the coherence ratios are estimated from the same runs.

Held-out validation. We checked this procedure on causal GPTs with the initialization above, tied embeddings, and BF16 storage with FP32 normalization, softmax, residual additions, and loss arithmetic, using 128-token sequences, batch size four, and a 4,096-token vocabulary. The ideal computation is evaluated in FP64 from the unrounded parameter draws, so storage casts contribute to the local errors. For each of two shapes, width 128 with depth 12 and width 256 with depth 24, 16 calibration initializations determine the constants and 16 new initializations form the held-out group. A depth-transfer group of 8 models of width 128 and depth 48 and an independent-label group of 4 models of width 128 and depth 12, whose targets are drawn independently of the inputs, are evaluated with the depth-12 constants. Calibration multiplies every fitted moment by 1.25, an empirical margin rather than a confidence guarantee, and chooses $\bar{c}$ and l_0 to minimize $\Gamma/(2-\bar{c})$ subject to $l_0 \le L/2$, which gives $\bar{c} \approx 1.2$–1.4 and $l_0 = 6$–8. The profile bound uses the second moments of the evaluated models and the fitted factors; the parameter-gradient bound (6) uses the fitted coherence ratios and the forward factor error from the profile bound, and the *measured-factor bound* supplies the measured α_A and α_B instead.

Every bound in Table 4 covered the RMS error of every group at every layer, with predicted relative errors of the parameter gradients below 11%. The forward and backward bounds are within small

Table 4: Ratio of each bound to the observed RMS error. At each layer, squared norms are averaged over the models of a group before normalization; entries give the minimum and maximum of the resulting ratio over layers. W2 is the second FFN weight matrix and QKV the attention input projection; the last row covers both.

	Held-out		Depth transfer	Independent labels
Width × depth	128 × 12	256 × 24	128 × 48	128 × 12
Models	16	16	8	4
Forward, profile bound	1.2–2.6	1.3–3.0	1.3–3.8	1.3–2.6
Forward, closed-form bound	15–19	15–22	15–24	15–19
Backward rounding	1.2–3.0	1.3–4.0	1.3–4.5	1.2–2.0
W2 gradient	3.1–6.5	3.9–7.9	3.9–11.5	2.1–4.2
QKV gradient	2.5–4.3	3.2–5.6	3.3–7.5	1.8–3.5
Measured-factor bound	1.6–2.0	1.6–2.2	1.5–2.4	1.1–1.7

factors of the observations; the parameter-gradient bounds are looser, mostly through the predicted factor errors, as the measured-factor row shows. Using the second moments of the calibration runs in the profile bound changes the forward ratios by less than 2%. The closed-form bound is looser mainly because its envelope must cover the spread of $N_l/(l+1)$ over layers. Bounding the coherence ratios by K^2M instead of using their fitted values gives up to 124 times the observed error.

Hypothesis checks. The fitted constants are hypotheses and can be checked on each group. The coherence ratios, formed from the moments of the held-out and transfer groups, stay within the calibrated limits. On the depth transfer, the conditioning factor K and the constant L_B of Assumption 6 exceeded their calibrated limits by up to 20%, so the parameter-gradient bounds there show empirical coverage with transferred constants rather than an application of Theorem 9. With independent labels, the W2 conditioning factor and coherence ratios were several times their calibrated limits, because targets independent of the inputs make the gradient itself less coherent across tokens. Agreement of a bound with observations therefore does not show that its fitted constants transfer to a new data policy or depth range, and rechecking these ratios indicates when a configuration needs recalibration.

E.9 Interpretation and scope

The main-text descriptions summarize several distinct roles in the analysis. Residual second moments supply the normalization for relative errors, perturbation bounds control amplification, and rounding-error moments determine how new discrepancies add to those already present. The proxy comparison then uses the common Jacobians of two linearized computations without requiring them to be evaluated explicitly. The following correspondence collects the precise conditions used for these steps.

Main-text description	Formal requirement
Residual growth and LayerNorm variance	Lemma 4 with its signal condition, and the denominator condition in Lemma 5.
Moderate amplification and cancellation	Mean-square gains $1 + c'_l/l$ with $c'_l \le \bar{c} < 2$ from layer l_0 on, and the local cross-moment bounds in Assumptions 3 and 4.
Comparable gradient and error amplification	Bounded interval products of the relative gains μ_k in Assumption 4.
Limited cancellation and norm fluctuations	Bounded coherence ratios in Theorem 9, implied by the conditioning and fourth-moment bounds of Assumption 5, together with Assumption 6.
The proxy covers relevant error directions	Assumption 7 and the first-order expansion in Lemma 10, with remainder control as in (10).
FP8 scaling factors from the full tensor	Scaling factors of the candidate and the reference that are computed from the same tensor or related by a power of two, so that the marginal moments of the local differences at quantizers follow Lemma 12 under its phase hypotheses; the directional comparison of Assumption 7 remains a separate condition (Appendix E.7).

The initialization establishes vanishing expected residual cross terms and, together with the signal condition, the second-moment bounds, even when sequence positions are dependent. It does not establish a lower bound on every realized normalization denominator, nor does it determine the joint moments of rounding errors generated by particular kernels. Those additional conditions enter through the perturbation and rounding assumptions, whose constants describe the architecture and arithmetic configuration being analyzed.

Probabilistic rounding analyses provide the motivation for mean-square error bounds under suitable moment conditions [16, 17]. For deterministic round-to-nearest, the result of each operation is fixed by its operands, so the conditional second moments used here belong to an explicit statistical model. Stochastic rounding can supply conditionally mean-zero errors for individual operations [5], but reuse of errors and nonlinear computations can produce aggregate bias [8, 10]. Accordingly, the assumptions are imposed on the local errors that appear in the recurrences and on the grouped differences that appear in the proxy comparison.

Results concerning the gradient independence assumption likewise depend on the architecture and initialization [58, 60]. Our backpropagation bound compares amplification of gradients and their errors, and the parameter-gradient bound uses conditioning and norm moments, so neither requires independent sequence positions or independent activation and gradient factors.

The accumulation analysis extends beyond the initialized Pre-LN model when the corresponding growth and moment conditions hold. The growth constants of Lemmas 4 and 5 are derived for the Pre-LN GPT block; for architectures with RMSNorm, gated feed-forward branches, or state-space mixers, the accumulation theorems apply with constants that are assumed rather than derived, and the tolerance estimate rests on the proxy comparison of Appendix E.6. The proxy comparison applies to other architectures when the augmented computation has a Lipschitz Jacobian and the local second-moment conditions hold; for FP8 recipes, common grids support the local model of Appendix E.7, and other scaling schemes require their own local second-moment comparison. The Taylor expansion requires aligned discrete routing decisions, which nnTrace enforces through rewriting.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The claims in the abstract and introduction accurately reflect the paper's contributions and scope.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Section 5.3 discusses false negatives and the scope of the false-positive results; Appendix C.2 states the scope of the safety-margin calibration and its fault model; Appendix E.9 states the scope of the theory.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

Answer: [Yes]

Justification: Appendix E states the assumptions with each result and gives complete proofs; Section 4 summarizes them.

Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: The method and information needed to reproduce this work are all disclosed.

   Guidelines:

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

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

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

Answer: [Yes]

Justification: The code is provided.

Guidelines:

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

6. **Experimental setting/details**

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

Answer: [Yes]

Justification: The perturbation scales, the safety margin and its calibration, the models, initialization, data, and FP8 recipe are specified in Section 3.2 and Appendix C.

Guidelines:

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

7. **Experiment statistical significance**

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

Answer: [No]

Justification: The tolerance results are reported as distributions and maxima over all traced tensors (Appendix C.2), including the variation over perturbation seeds, rather than as means with error bars; the held-out validation of the bounds in Appendix E.8 uses separate calibration and evaluation initializations.

Guidelines:

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

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: Appendix C.1 gives the hardware and software for each group of experiments; Section 5.1 and Appendix C.4 give the detection and tracing times.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The work conforms with the code of ethics.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [N/A]

    Justification: There is no societal impact of the work performed.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

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

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: The assets are cited and provided with version used.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

Answer: [N/A]

Justification: The paper does not release new assets.

Guidelines:

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

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

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

Answer: [N/A]

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

Guidelines:

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

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

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

Answer: [N/A]

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

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

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

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.