ATTNChecker: Highly-Optimized Fault Tolerant Attention for Large Language Model Training

Yuhang Liang[1], Xinyi Li[2], Jie Ren[3], Ang Li[2], Bo Fang[2], Jieyang Chen[1]

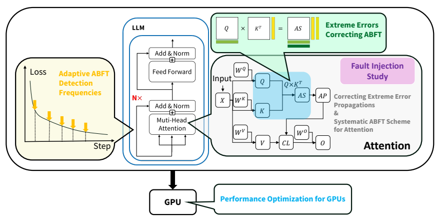
[1]University of Oregon, [2]Pacific Northwest National Laboratory, [3]William & Mary

1. Introduction

- Training a large language model (LLM) is an extremely time and resource-intensive process, often taking weeks or even months to reach the desired accuracy.
- Recent investigations showed that the training of LLMs has high sensitivity of soft errors. The soft errors can bring the negative impact to training accuracy, even interrupt the training process.
- Current techniques, such as checkpointing, cannot efficiently handle the soft error during LLM training.
- ATTNChecker is a ABFT technique tailored for the attention mechanism in LLMs. It is designed based on fault propagation patterns of LLM and incorporates performance optimization to adapt to both system reliability and model vulnerability, while providing lightweight protection for fast LLM training.

2. ATTChecker Overview

- **Extreme Errors Correcting ABFT:** Handle the INF, Nan, near-INF during LLMs training, whereas the standard ABFT cannot efficiently handle these errors.
- **Systematic ABFT Scheme for Attention:**
  - We divide the attention into three protection sections and allow the errors to propagate up 1D (1 column or 1 row).
  - We differentiate the **Deterministic** and **Nondeterministic** patterns and apply different strategies to handle the errors.
    - **Nondeterministic:** Use both column and row checksums.
    - **Deterministic:** Use single-sided checksum.
- **Performance Optimizations:**
  - Customized and efficient kernels for encoding checksums.
  - Customized error detection and correction kernels for GPU to optimize the detection frequencies

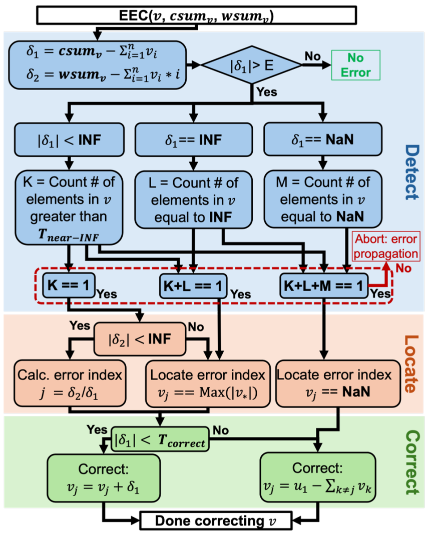

LLM
Add & Norm
Feed Forward
Add & Norm
Muti-Head Attention
Loss
Step
Adaptive ABFT Detection Frequencies
Extreme Errors Correcting ABFT
Fault Injection Study
Input
Q
K
V
$Q \times K^T$
AS
AP
CL
O
W^Q
W^K
W^V
W^O
Correcting Extreme Error Propagations & Systematic ABFT Scheme for Attention
Attention
GPU
Performance Optimization for GPUs

3. Fault Injection Study

- **Experiment**: Randomly select 10% (~5,000) elements of each output matrix to inject one fault.
- **Findings:**
  - **Error Patterns in Attention**:
    - Errors can quickly propagate with diverse types of chances.
    - Earlier injected faults lead more negative impact to LLM training.
  - **Loss**:
    - 2D pattern leads to the non-correctable state.
    - Q, K and V have higher fault sensitivity.

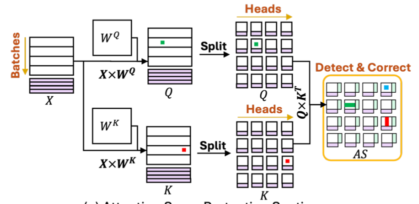
4. Extreme Errors Correcting ABFT

- EEC-ABFT first compares the updated checksums $csum_v$ and $wsum_v$ with recompute checksum to get the difference δ_1 and δ_2 and uses δ_1 to detect the error.
- Upon detecting an error, instead of relying on a unified error handling strategy, EEC-ABFT applies corresponding error locating and correction procedures for 4 different cases by using δ_1.

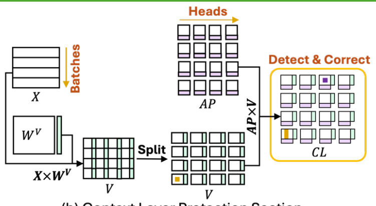

EEC(v, $csum_v$, $wsum_v$)
$\delta_1 = csum_v - \Sigma_{i=1}^{n} v_i$
$\delta_2 = wsum_v - \Sigma_{i=1}^{n} v_i * i$
$|\delta_1| > E$
No
No Error
Yes
$|\delta_1| <$ INF
$\delta_1 ==$ INF
$\delta_1 ==$ NaN
K = Count # of elements in v greater than $T_{near-INF}$
L = Count # of elements in v equal to INF
M = Count # of elements in v equal to NaN
Abort: error propagation
K == 1
K+L == 1
K+L+M == 1
Yes
No
$|\delta_2| <$ INF
Calc. error index $j = \delta_2/\delta_1$
Locate error index $v_j ==$ Max($|v_*|$)
Locate error index $v_j ==$ NaN
$|\delta_1| < T_{correct}$
Correct: $v_j = v_j + \delta_1$
Correct: $v_j = u_1 - \Sigma_{k \neq j} v_k$
Done correcting v
Detect
Locate
Correct

5. Systematic ABFT Scheme for Attention

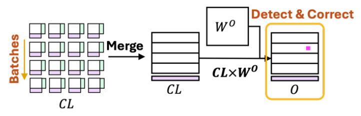
- We divide the attention into three protection sections, shown in (a), (b) and (c).
- In each section, we allow errors propagate up to 1D and handle these 1D errors to reduce the overhead (e.g. an error occurring in Q is handled at the Attention Scores (AS)).
- Attention Scores (AS) and Context Layer (CL) are nondeterministic. as both 0D and 1D errors can occur in these matrices. To effectively handle such errors, we apply both column and row checksums.
- Attention Output (O) is deterministic, since only 0D error could occur in this matrix. Therefore, we use single-side checksum (column checksum) for error detection and correction.



(a) Attention Score Protection Section



(b) Context Layer Protection Section

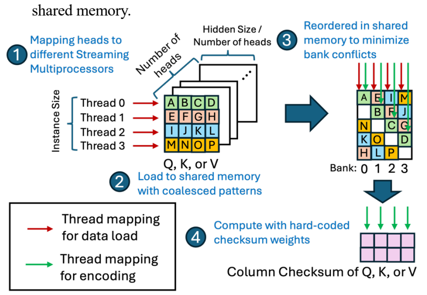


(c) Attention Output Protection Section

6. Performance Optimizations

- **Customized and efficient kernels for encoding checksums**

  We parallelize the encoding process along the Streaming Multiprocessor by the Number of Heads ×Number of Batches to improve GPU occupancy. To encode each block, we load all data into shared memory and decoupled thread data mapping between loading and computing to achieve fully coalesced global memory accesses while minimizing bank conflict in shared memory.


1 Mapping heads to different Streaming Multiprocessors
Number of heads
Hidden Size / Number of heads
Instance Size
Thread 0
Thread 1
Thread 2
Thread 3
A B C D
E F G H
I J K L
M N O P
Q, K, or V
2 Load to shared memory with coalesced patterns
3 Reordered in shared memory to minimize bank conflicts
Bank: 0 1 2 3
4 Compute with hard-coded checksum weights
Column Checksum of Q, K, or V
Thread mapping for data load
Thread mapping for encoding

- **Frequencies Optimization Algorithm**

  We treat the probability distribution of the number of error occurrences as Posson distribution, and used a greedy algorithm, shown in Algorithm 1, to optimize the ABFT frequency.

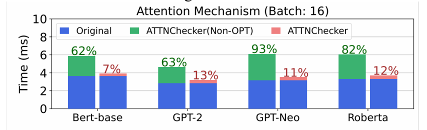
Algorithm 1: ABFT Frequencies Optimization

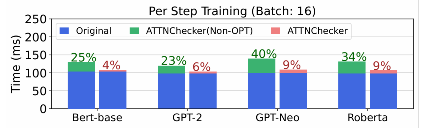
```
Function OptimizeABFTFrequencies($\lambda^{INF}, \lambda^{NaN}, \lambda^{nINF}, \phi, FC_{target}$):
    $FCE_{AS}, FCE_{CL}, FCE_{O} \leftarrow \lambda^{INF}, \lambda^{NaN}, \lambda^{nINF}, \phi$
    $FC \leftarrow 0$ ; $t_{AS}, t_{CL}, t_{O} \leftarrow 0$
    for $S$ in $DescendSort(FCE)$ do
        if $FC_{target} - FC < FCE_S T_S$ then
            $t_S = T_S$
        end
        if $FC < FC_{target}$ then
            $t_S = \frac{FC_{target} - FC}{FCE_S}$
        end
        $FC \leftarrow FC + FCE_S t_S$
    end
    $f_{AS} \leftarrow \frac{t_{AS}}{T_{AS}}$ ; $f_{CL} \leftarrow \frac{t_{CL}}{T_{CL}}$ ; $f_O \leftarrow \frac{t_O}{T_O}$
return $f_{AS}, f_{CL}, f_O$
```

7. Results

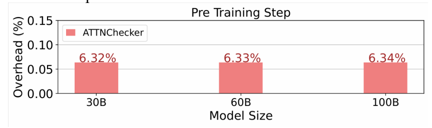
- **ATTNChecker Overhead**

  We compare ATTNChecker with the non-optimized version of ATTNChecker, which applies ABFT to all GEMMs in the attention. ATTNChecker only brings 11% overhead to attention and 7% overhead to training.



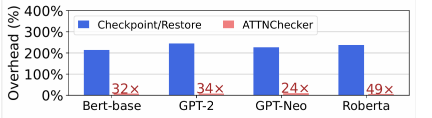


The estimated overhead of ATTNChecker during large-scale training is approximately 6.3% and remains nearly constant as the number of parameters increases.



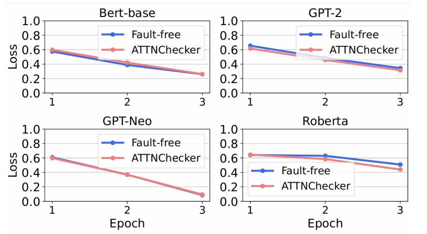
- **Recovery Overhead**

  Assuming the checkpointing is performed at every training step. And Checkpointing/Restore causes more than 200% overhead, while ATTNChecker achieving 49X speed up.



- **Loss after 3-Epoch Training**

  The losses between baseline and ATTNChecker after 3-epoch training shows that ATTNChecker makes negligible impact to loss



8. Conclusion

We introduce ATTNChecker, an ABFT technique that offers a lightweight, real-time solution for detecting and correcting errors during LLM training. ATTNChecker is efferent in enhancing the resilience and efficiency of LLM training and provides a robust framework for future fault tolerance in large-scale machine learning models.

Published at ACM PPoPP'25

Liang, Yuhang, Xinyi Li, Jie Ren, Ang Li, Bo Fang, and Jieyang Chen. "ATTNChecker: Highly-Optimized Fault Tolerant Attention for Large Language Model Training." In Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, pp. 252-266. 2025.