Connecting the Impact of Silent Data Corruption with Different Training Characteristics: An Empirical Study

Hengzhi Pei, Leonard Lausen, and George Karypis, *Amazon Web Services, Santa Clara, CA, 95054, USA*

Despite the nonnegligible occurrence of silent data corruption (SDC) during large-scale training of large language models (LLMs), SDC impact on training lacks systematic understanding. This article empirically analyzes the connections between different training characteristics and the impact of SDC on LLM training. Using deterministic training workloads on real-world SDC-affected hardware, we quantify SDC impact by measuring the difference from the baselines on healthy hardware and provide insights into training robustness against SDC by systematically controlled experiments. We find that SDC impact correlates strongly with training stability and loss landscape regions, with not-a-number (NaN) occurring during training larger models. We further study if setting elementwise gradient bounds can mitigate SDC impact considering that SDC can change gradients by large magnitudes. Our results show that a proper bound can reduce the relative performance gap caused by SDC but cannot avoid SDC-induced NaN and may compromise training performance on healthy hardware.

Silent data corruption (SDC) is an increasingly important challenge for hardware where faulty components in CPUs or machine learning accelerators can silently produce erroneous results without notification. Such corruption can occur due to specific conditions, like temperature and device characteristics,[1] which will negatively affect the corresponding workloads and service. Although some hardware-based techniques, like error correction code for SRAM and parity checks for networks, are applied to reduce the error rate, not all the hardware blocks are protected from SDC and it usually requires extensive diagnostics to detect an SDC-affected node.

Large distributed training jobs, like pretraining a large language model (LLM), commonly involve thousands of accelerators over long periods, which greatly increases the probability of SDC occurrence as reported across the industry.[2,3] During the lifespan of such jobs, some originally healthy nodes can degrade due to aging and latent defects, and begin to produce SDCs. These degraded nodes may remain undetected until comprehensively stress-tested. Wrong computational results from SDC can thus silently affect LLM training, which raises concerns about the quality of the resulting model. Previous studies have shown that SDC can lead to accuracy degradation and even training divergence.[4,5]

In this article, we comprehensively investigate the impact of SDC on training from machine learning perspectives. We aim to establish connections between the impact of SDC and intrinsic characteristics of the training workload. To achieve this goal, we collected real-world nodes exhibiting SDC and executed various deterministic LLM pretraining workloads by leveraging an XLA compiler. This allowed us to compare the training between healthy and unhealthy nodes while attributing any observed differences to SDC, thereby helping us understand the impact of SDC.

By empirical evaluation, we connect SDC impact with the following training characteristics: 1) training stability, 2) loss region, and 3) model size. We carefully designed experiments for each factor to avoid

Digital Object Identifier 10.1109/MM.2025.3642709

confounding effects. Our evaluation shows that the impact of SDC is more severe when training is more unstable. We also found that SDC has different degrees of impact on different loss regions, even when training is stable. Furthermore, training a larger model with a larger hidden size makes it not only easier to trigger SDC but can also yield not-a-number (NaN) results when training runs on certain unhealthy nodes. Motivated by the prior work's finding that SDC can change gradients by a large magnitude,[5] we further consider if setting gradient bounds can help mitigate the impact of SDC. We experimented with absolute bounds and spike-aware bounds from previous work[6] that apply element-wise bounds on gradients to study the connection between the impact of SDC and the gradient value. We found that gradient bounds, especially spike-aware gradient bounds, can reduce the relative performance gap between the healthy node and unhealthy nodes. But such benefit requires careful selection of the bounding threshold and may negatively impact training loss on the healthy node. We further found that gradient bounds cannot solve the NaN risk caused by SDC when training a large model. Our findings provide crucial insights into the robustness of LLM training against SDC, shedding light on the future mechanisms to mitigate the impact of SDC on training.

METHODOLOGY

In this section, we introduce our methodology to investigate how different training characteristics relate to SDC impacts. We first accessed SDC-affected hardware in the real world and then designed different deterministic workloads for end-to-end comparisons on the training performance across different nodes.

Hardware With SDC

We followed a methodology established by prior work[5] to obtain *unhealthy* nodes affected by SDC from a cloud provider. Each node is a Linux server with non-uniform memory access architecture containing multiple machine learning accelerators for high-performance deep learning applications. In our study, an unhealthy node failed at least one of the production stress tests from the provider. The production stress tests run deterministically on the accelerators and include communication collective tests, matrix multiplication tests, and a model training run where training outputs are compared with precomputed ground-truth values. When the execution on one node leads to a different value from the ground-truth or to nondeterminism over multiple runs, the node is marked as unhealthy, indicating the existence of SDC. We note that not all workloads running on an unhealthy node would experience SDC, but only certain workloads can be affected by SDC. Meanwhile, different unhealthy nodes are affected by SDC to varying degrees.[5] Correspondingly, *a healthy node* passes all of the aforementioned tests, which means that no workload on such a node will experience SDC. In total, we used 12 unhealthy nodes in our experiments and we denote them as node 1 to node 12.

Workload Setup

We employed an XLA compiler, which compiles the training code from popular frameworks like PyTorch for high-performance execution, to run the exactly same training workload with the same random seeds on each single node. In this way, we can isolate other possible sources of nondeterminism and randomness because the XLA compiler can ensure fully deterministic execution. Consequently, the computation results of a workload are the same across healthy nodes, allowing any difference between healthy and unhealthy nodes to be attributed solely to SDC.

We focused on pretraining LLMs, which is the predominant application of large-scale training, to understand this topic in more realistic scenarios. LLM pretraining workloads involve training a large decoder-only Transformer model on massive text corpora. The large model size not only necessitates the distributed computation across accelerators, but also makes training sensitive to various hyperparameters that can significantly affect training dynamics and stability.[7] To maintain training stability, Adam optimizer is widely used with gradient norm clipping and a learning rate warm-up schedule where the learning rate gradually increases to a target value during the warm-up stage.

Unless otherwise specified, in our study we used a decoder-only Transformer model with hidden size as $H = 2304$ and $L = 24$ layers following previous settings.[8] We chose this model size to strike a balance among the computation efficiency, the sensitivity to training characteristics, and the visibility of SDC impacts. We conducted mixed-precision training on each single node for 5000 steps by default. We used tensor parallelism to shard the computation of each layer and used data parallelism to replicate the model on multiple sets of accelerators within a single node. We used a ZeRO-1 optimizer to shard the optimizer states across all accelerators. We processed the C4 dataset as the training dataset where the batch size is 512 and each sample has a sequence length of 2048 tokens. We used AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$ and used

gradient clipping at global norm 1. The target learning rate was $LR = 5e{-}4$ and the default learning rate warm-up step was 1750. This single-node setup can serve as a proxy for the real-world LLM pretraining workloads at larger scales[7] and allowed us to comprehensively study how different training characteristics correlate with the impact of SDC.

Metrics

Starting from this default setup, we systematically controlled different training characteristics to analyze their relations with SDC impacts by observing the end-to-end training difference across unhealthy nodes. To quantify the impact of SDC, we report the following two metrics:

- *Worst case:* We calculated the maximum final training loss over unhealthy nodes and also report the percentage of the loss increase relative to the performance on a healthy node. If training diverges or NaN happens, we report the number of such unhealthy nodes under this workload.
- *Number of worse cases:* We define training on an unhealthy node is significantly worse than that on a healthy node under the same training setup if the percentage of the final loss increase is larger than , i.e., $L_{\text{unhealthy}} - L_{\text{healthy}}/L_{\text{healthy}} > 0.5\%$.

Qualitatively, we also examined the training trajectories.

CONNECTION WITH TRAINING STABILITY

Experiment Design

First, we studied the connection between training stability and SDC impact. We used two important training hyperparameters to control the training stability: *learning rate* (*LR*) and β_2 in Adam optimizer. The learning rate controls the size of the parameter updates during training and β_2 controls the exponential moving average of the second momentum in Adam optimizer. Previous studies show that training with a large learning rate or a large β_2 can lead to training instability.[7,8] In our experiments, we evaluated $LR \in \{5e-4, 1e-3, 5e-3\}$ while fixing $\beta_2 = 0.95$ and evaluated $\beta_2 \in \{0.99, 0.999\}$ while fixing $LR = 5e{-}4$.

Results

Table 1 shows the quantitative results for SDC impact under different training setups measured by the worst case and the number of worse final training loss. We found that when training is stable (e.g., $LR = 5e{-}4$, $\beta_2 = 0.95$), SDC has minimal impact on final training loss,

TABLE 1. Quantitative results for the impact of SDC under different training settings with different *LR* and β_2.

Setting	*LR*	β_2	Healthy Loss	Worst Case	No. of Worse
From scratch	5e–4	0.95	2.6745	2.679 (+0.17%)	0
	1e–3	0.95	2.635	2.6434 (+0.32%)	0
	5e–3	0.95	2.6133	diverge (1)	5
	5e–4	0.99	2.7042	2.7114 (+0.27%)	0
	5e–4	0.999	2.7578	2.8428 (+3.08%)	6
From checkpoints	5e–3	0.95	2.6071	2.6085 (+0.05%)	0
	2e–3	0.99	2.6854	2.7179 (+1.21%)	5

When training from scratch, we train 5000 steps; When continuing training from checkpoints, we train until 6000 steps.

with unhealthy nodes achieving similar or even better performance than that on a healthy node. However, as training becomes more unstable with a larger learning rate or β_2, SDC increasingly degrades final training performance and can even cause training divergence. Figure 1 shows the training trajectories across different nodes when changing the learning rate and β_2, respectively. As the learning rate or β_2 increases, training on a healthy node becomes less stable and exhibits more loss spikes. Consequently, SDC exacerbates this instability and leads to even more erratic training behavior on unhealthy nodes. When $LR = 5e{-}3$, although the final training performance on the healthy node is the best compared to lower learning rates, there will be more severe loss spikes during training on unhealthy nodes and training on Node 8 even diverges after a loss spike at around 2000 steps. It demonstrates that the impact of SDC on the training trajectory is more pronounced when training is less stable.

CONNECTION WITH LOSS REGION

Experiment Design

Next, we studied the connection between the impact of SDC and different regions of the loss landscape. The warm-up stage, where the learning rate gradually increases to the target value, can be more unstable than later training phases because the loss landscape is sharper at the early stage.[9] As a result, SDC can lead the model to substantially different local minima during the warm-up stage. Therefore, we wanted

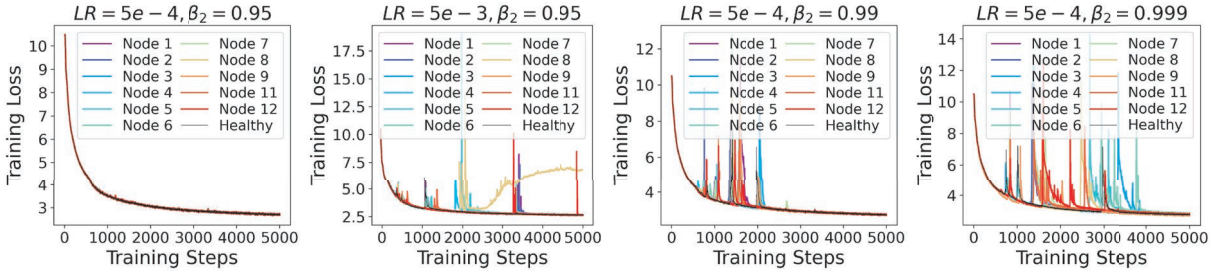

FIGURE 1. Training curves with different learning rates and β_2 on different nodes. Note that node 10 does not show nondeterminism under these workloads.

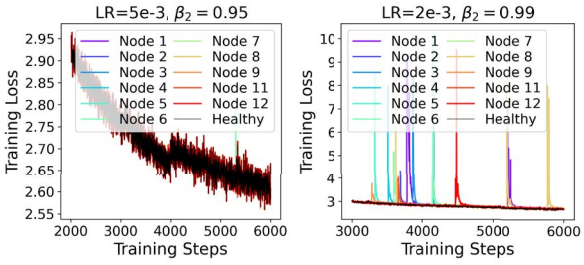

FIGURE 2. Training curves with different training hyperparameters on different nodes when loading from the checkpoints from a healthy node. Note that node 10 does not show nondeterminism under these workloads.

to further study how SDC would affect training when starting from the same post-warm-up training state.

We continued training from the checkpoints pretrained on a healthy node to understand the impact of SDC on the subsequent training. To mitigate the influence of training stability, we first continued training from these checkpoints on a healthy node to validate the target training period on the healthy node is stable, and then performed the same training workloads on unhealthy nodes. In our experiments, we used the following two settings to get the checkpoints from a healthy node:

- *Setting 1:* We used the checkpoint at 2000 steps trained with $LR = 5e{-}3$ and $\beta_2 = 0.95$.
- *Setting 2:* We used the checkpoint at 3000 steps trained with $LR = 2e{-}3$ and $\beta_2 = 0.99$.

For both settings, we restored the model and optimizer states from the respective checkpoints and continued training until 6000 steps.

Results

Table 1 shows the quantitative results for SDC impacts on continuing training from checkpoints. Figure 2 shows the corresponding training trajectories across different nodes. Under the first training setup with $LR = 5e{-}3$, $\beta_2 = 0.95$, the training curves on unhealthy nodes are generally similar to that on the healthy node. We only see a small loss spike on node 6 and the worst final training loss on unhealthy nodes is also comparable to that on the healthy node. However, as shown in the previous section, SDC would lead to training divergence on unhealthy nodes when we train from scratch with the same training hyperparameters. By comparing the results of training from scratch and training from a postwarm-up checkpoint, we conclude that SDC exhibits larger impacts during the warm-up stage of training.

Under the second training setup with $LR = 2e{-}3$, $\beta_2 = 0.99$, more severe loss spikes happened across many unhealthy nodes and the relative loss increase in the worst case was nonnegligible, although this training period was stable without significant loss spikes on the healthy node. The difference is possibly because training with a large β_2 leads the training trajectory to a sharper loss region where SDC is more likely to cause loss spikes, which demonstrates that the impact of SDC is closely related to the loss landscape.

CONNECTION WITH MODEL SIZES

Experiment Design

In our preliminary study, we found that the size of the training workload affects the occurrence of SDC. For example, on node 6, we found that SDC does not happen when we train a model with hidden size of 1024 but happens when we set the hidden size as 2048. It is possibly because the smaller models either fail to activate the faulty units on the hardware or do not achieve sufficient system utilization to trigger SDC events. However, it remains unclear if SDC would have larger impacts on training larger models when all model sizes are large enough to trigger SDC. Therefore, we further compared training the default model with two larger models:

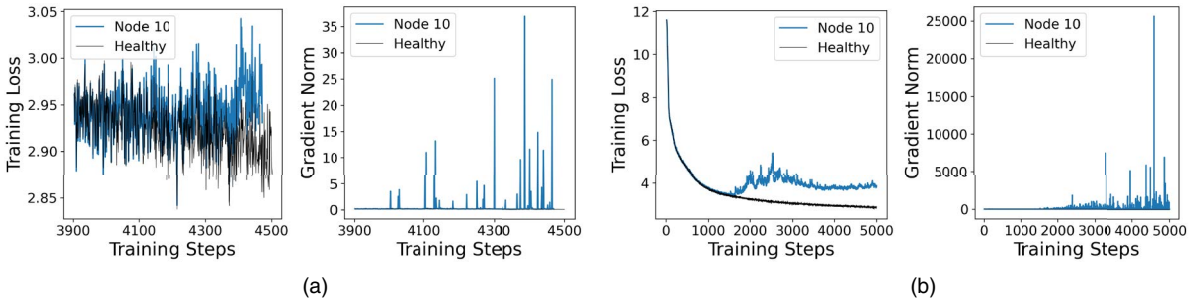

FIGURE 3. (a) Training model 2. Training curves for a large model ($H = 6144$, $L = 12$) on node 10 and on a healthy node. (b) Training model 2 (skip the steps with Nan). We conducted another training run on node 10, skipping the training steps where NaN happens.

- *Model 1:* $L = 24$ layers with hidden size $H = 4096$.
- *Model 2:* $L = 12$ layers with hidden size $H = 6144$.

Since training larger models is naturally more unstable (e.g., more sensitive to large learning rates[7]), we needed to disentangle the effect of training stability to further study the connections between model sizes and the impact of SDC. For efficiency, we set the batch size to 256 and used the default hyperparameters $LR = 5e{-}4$, $\beta_2 = 0.95$. We verified that this training setting leads to stable training trajectories without significant loss spikes on the healthy node for all models.

Results

We found that pretraining of all three models leads to similar final training loss curves across most of unhealthy nodes and the relative increase of the final training loss is no larger than 0.13%. The only exception was that NaN emerges during pretraining model 2 on node 10. Figure 3(a) shows the detailed training trajectories before NaN happens on node 10. We found training on this unhealthy node begins to produce very large gradient spikes and the training loss shows a gradual increasing trend before NaN happens. On this same unhealthy node, we repeated these three workloads and confirmed that training the default model does not exhibit any SDC and training the model 1 yields a similar training trajectory to that on the healthy node. This difference suggests that training with a model with larger hidden sizes can amplify SDC impacts even when training is stable. There could be two possible reasons. First, training a larger model would have higher accelerator utilization, which plausibly increases SDC occurrence. Second, numerical issues can happen more easily during the training of larger models, as some computational results can become too large or too small, making them more susceptible to being corrupted into NaN by SDCs.

We further checked whether simply skipping the training steps where NaN happens (in either forward or backward computation) could make training comparable to that on a healthy node. Figure 3(b) shows the corresponding training trajectory on node 10, and we found that training diverges after 1500 steps. This indicates that skipping the steps with NaN cannot eliminate the impact of SDC. This is expected because skipping the steps with NaN does not solve the issue of gradient explosion witnessed in Figure 3, which eventually prevents training convergence.

CONNECTION WITH GRADIENT VALUE

Experiment Design

Gradients of weights are crucial in model training. Previous studies report that SDC can alter some elements in the gradients by large magnitudes,[5] which would potentially corrupt the model quality. Motivated by activation bounding techniques used during inference to defend against bit-flip errors,[10] we further investigated if setting a bound to each gradient value g_i can mitigate the impact of SDC on training. While gradient norm clipping has been used, it fails to prevent training divergence caused by SDC, as shown in the previous section, because a gradient outlier polluted by SDC can still significantly change the optimization direction.

Therefore, we evaluated two *element-wise* gradient bounding methods: 1) *Absolute bound* clips all gradients when their absolute value exceeds a threshold θ_{abs}. We set $\theta_{abs} \in \{2e-4, 1e-3, 5e-3\}$; 2) *Spike-aware bound* proposed in the previous work[6] clips the gradient value that causes a gradient spike based on the following criterion:

$$g_i = \text{sign}(g_i)\sqrt{\theta_{\text{spike}} V_i} \text{ if} \frac{g_i^2}{V_i} > \theta_{\text{spike}} \quad (1)$$

where θ_{spike} is the spike-aware bound threshold and V_i is the second momentum in Adam optimizer, which calculates the moving average of g_i^2. Previous work

suggests that the gradient spikes can be harmful to LLM training.[6] Therefore, applying this bound may simultaneously improve LLM training on a healthy node and mitigate the impact of SDC. In our experiment, we set $\theta_{\text{spike}} \in \{10k, 1k, 100\}$.

To validate their effectiveness for mitigating the impact of SDC, we conducted experiments under three settings where SDC demonstrates significant impacts based on the results from the previous sections:

- training from scratch with $LR = 5e{-}3$, $\beta_2 = 0.95$
- training from a checkpoint at 3000 steps with $LR = 2e{-}3$, $\beta_2 = 0.99$
- training a model with $H = 6144$, $L = 12$ from scratch using $LR = 5e{-}4$, $\beta_2 = 0.95$ on node 10 where NaN happens.

Results

Table 2 shows the quantitative results for SDC impact when using different gradient bounds during training from scratch. Figure 4 shows representative training curves with different gradient bounds across different nodes. We first found that setting a fixed absolute bound threshold is often insufficient to avoid training divergence when using $\theta_{\text{abs}} = 5e{-}3$ and $\theta_{\text{abs}} = 1e{-}3$. Although using $\theta_{\text{abs}} = 2e{-}4$ can avoid training divergence on unhealthy nodes, it leads to a significant increase in the training loss on the healthy node. By contrast, the spike-aware bound allows more adaptive ranges for different gradient elements, which can prevent training divergence without harming the training performance on the healthy node too much. It demonstrates that spike-aware bounds are more effective in mitigating SDC impacts than absolute bounds.

TABLE 2. Quantitative results for the impact of SDC when using different gradient bounds during training from scratch.

Bound	Threshold	Healthy Loss	Worst Case	No. of Worse
None		2.6133	Diverge (1)	5
Absolute	2e–4	2.6711	2.6691 (–0.07%)	0
	1e–3	2.6154	Diverge (1)	4
	5e–3	2.6551	Diverge (1)	1
Spike-aware	100	2.6293	2.6511 (+0.83%)	1
	1000	2.6047	2.6331 (+1.09%)	5
	10,000	2.6178	2.657 (+1.50%)	3

Table 3 shows the quantitative results for SDC impacts when using different gradient bounds during continuing training from a checkpoint. Similarly, although the gradient bounds with proper thresholds can reduce the performance gap between the healthy node and unhealthy nodes, they still more or less make the training performance on healthy nodes worse. It is possibly because large gradient values are important during LLM training since the gradient distributions are heavy-tailed.[11] As a result, further clipping those values may slow down the convergence rate, particularly when gradient norm clipping is already used. Under this setting, spike-aware bounds are also more effective in mitigating the impact of SDC. Figure 5 shows the training curves across different nodes when using the best absolute bound ($\theta_{\text{abs}} = 5e{-}3$) and the best spike-aware bound ($\theta_{\text{spike}} = 100$) in this experiment. Although using $\theta_{\text{spike}} = 100$ during training can make the loss spike occurrence across unhealthy nodes less frequent, it cannot fully alleviate the loss spikes on unhealthy nodes, which demonstrates the limited effectiveness of the gradient bounds.

For the third setting of training the large model ($H = 6144$, $L = 12$) with gradient bounds, we found that all gradient bounds fail to prevent the NaN issue that can happen both during forward and backward computation at different steps. It indicates that NaN found on

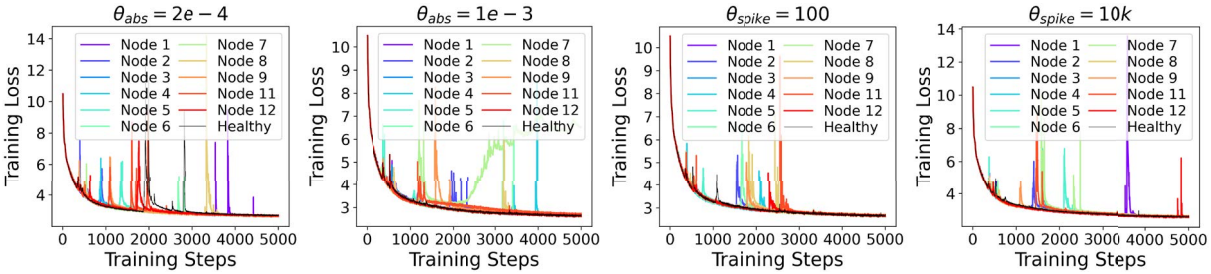

FIGURE 4. Training curves with two gradient bounding methods across different nodes when training from scratch.

TABLE 3. Quantitative results for the impact of SDC when using different gradient bounds during continuing training from a checkpoint.

Bound	Threshold	Healthy Loss	Worst Case	No. of Worse
None		2.6854	2.7179 (+1.21%)	5
Absolute	2e−4	2.7113	2.7227 (+0.42%)	0
	1e−3	2.7118	2.7107 (−0.04%)	0
	5e−3	2.6923	2.7033 (+0.41%)	0
Spike-aware	100	2.6861	2.6879 (+0.07%)	0
	1000	2.692	2.6963 (+0.16%)	0
	10,000	2.6976	2.7013 (+0.14%)	0

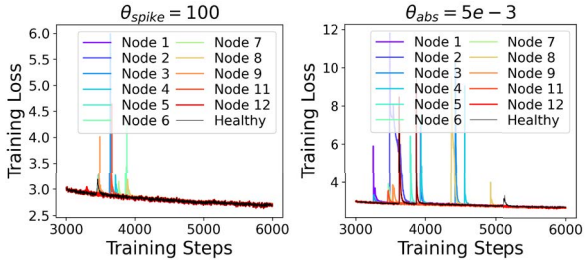

FIGURE 5. Training curves with the spike-aware bound ($\theta_{\text{spike}} = 100$) and the absolute bound ($\theta_{\text{abs}} = 5e{-}3$), respectively, across different nodes when continuing training from a checkpoint at 3000 steps.

node 10 is less relevant to training dynamics but more related to the properties of faulty units themselves.

In summary, gradient bounds cannot solve all the SDC-induced risks during LLM training and applying the gradient bounds in practice requires balancing faster convergence rate and training robustness against SDC.

CONCLUSION

In this article, we empirically reveal the connections between the impact of SDC and various characteristics of a training workload. We demonstrate that training instability, increased model hidden dimensions, and specific loss landscape regions can amplify SDC's detrimental effects on training. We also reveal that although setting gradient bounds can mitigate the impact of SDC to some extent, it may sacrifice the benign training performance and require careful selection of the threshold.

Our findings provide critical insights for enhancing the fault tolerance of large-scale training against SDC. As training an LLM with hundred billions of parameters can be naturally unstable, it is important to strike a balance between faster convergence rate and training stability in SDC-prone environments. Meanwhile, when an unusual training phenomenon happens, e.g., a loss spike or gradient explosion, it is important to ensure the health status of training nodes in case that SDC leads the training trajectory to diverge. Besides, given the complicated interplay between SDC and training optimization, regular deterministic tests to detect unhealthy nodes with SDC during training could be a more practical way to mitigate the impact of SDC.

On the other hand, we should also establish comprehensive benchmarks to better understand failure modes caused by SDC in production environments. For example, hardware providers could design diverse workloads to stress different pipelines within a machine learning accelerator at different levels and report the impact of SDC.

REFERENCES

1. H. Dattatraya Dixit et al., "Silent data corruptions at scale," 2021, *arXiv:2102.11245*.
2. R. Anil et al., "Gemini: A family of highly capable multimodal models," 2023, *arXiv:2312.11805*.
3. A. Dubey et al., "The Llama 3 herd of models," 2024, *arXiv:2407.21783*.
4. Y. He et al., "Understanding and mitigating hardware failures in deep learning training systems," in *Proc. 50th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2023, pp. 1–16.
5. J. Ma, H. Pei, L. Lausen, and G. Karypis, "Understanding silent data corruption in LLM training," in *Proc. 63rd Annu. Meeting Assoc. Comput. Linguistics (Volume 1: Long Papers)*, 2025, pp. 20,372–20,394, doi: 10.18653/v1/2025.acl-long.996.
6. T. Huang, Z. Zhu, G. Jin, L. Liu, Z. Wang, and S. Liu, "SPAM: Spike-aware Adam with momentum reset for stable LLM training," in *Proc. 13th Int. Conf. Learn. Representations*, 2025. Accessed: Mar. 2025. [Online]. Available: https://openreview.net/pdf?id=L9eBxTCpQG
7. M. Wortsman et al., "Small-scale proxies for large-scale transformer training instabilities," in *Proc. 12th Int.*

Conf. Learn. Representations, 2024. Accessed: Mar. 2025. [Online]. Available: https://proceedings.iclr.cc/paper_files/paper/2024/file/d848cb2c84f0bba7f1f73cf232734c40-Paper-Conference.pdf

8. S. Takase, S. Kiyono, S. Kobayashi, and J. Suzuki, "Spike no more: Stabilizing the pre-training of large language models," 2023, *arXiv:2312.16903*.
9. D. S. Kalra and M. Barkeshli, "Why warmup the learning rate? Underlying mechanisms and improvements," in *Proc. 38th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, 2024, pp. 111,760–111,801.
10. Z. Chen, G. Li, and K. Pattabiraman, "A low-cost fault corrector for deep neural networks through range restriction," in *Proc. 51st Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 1–13, doi: 10.1109/DSN48987.2021.00018.
11. J. Zhang et al., "Why are adaptive methods good for attention models?" in *Proc. 34th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, 2020, pp. 15,383–15,393.

HENGZHI PEI is an applied scientist at Amazon Web Service, Santa Clara, CA, 95054, USA. His research interests include natural language processing, adversarial machine learning, and distributed training. Pei received his M.S. degree in computer science from the University of Illinois Urbana-Champaign. Contact him at philepei@amazon.com.

LEONARD LAUSEN is a senior applied scientist at Amazon Web Service, New York, NY, 10001, USA. His research interests include machine learning, distributed training, and systems. Lausen received his M.Phil. degree in computer science and engineering from the Hong Kong University of Science and Technology. Contact him at lausen@amazon.com.

GEORGE KARYPIS is a Distinguished McKnight University Professor and William Norris Chair in large scale computing in the Department of Computer Science and Engineering at the University of Minnesota, Minneapolis, MN, 55455, USA, and a senior principal scientist at Amazon Web Service, USA. His research interests include data mining, learning analytics, and high-performance computing. Karypis received his Ph.D. degree in computer science from the University of Minnesota. He is a Fellow of the IEEE. Contact him at gkarypis@amazon.com.