International Journal of Scientific Research & Engineering Trends
Volume 12, Issue 5, Sep-Oct-2026, ISSN (Online): 2395-566X

Silent Data Corruption and the Case for Algorithmic Fault Tolerance

Faaiq Mushtaq
Computer Science Graduate, Kashmir University

Abstract- Silent data corruptions or SDCs are computational errors that produce a wrong but plausible result with no crash, no exception and no flag of any kind. For decades the working assumption in systems design was that a tested processor either computes correctly or fails loudly. Recent measurement studies from Meta, Google and Alibaba have overturned that assumption: mercurial cores that occasionally miscompute at rates far above what fault injection studies predicted are now a documented, recurring phenomenon in hyperscale fleets. This paper argues that the drivers behind SDCs, transistor density, lower operating voltages and sheer deployment scale, are structural rather than transient and that the problem will intensify over the next decade as workloads move toward quantized, low precision AI training and toward radiation exposed space computing. Because full hardware correction is prohibitively expensive and the space of possible fault sites is too large to fully screen, this paper surveys algorithmic fault tolerance, particularly checksum based approaches descended from classical ABFT, as a lightweight, mathematically grounded complement to hardware mitigation. It closes by proposing a concrete thesis direction: characterizing when a modest, bounded performance cost is worth paying to catch a corrupted core before its errors propagate through a distributed system or a multi week training run.

Keywords- Silent Data Corruption (SDC); silent errors; algorithm-based fault tolerance (ABFT); checksum; fault tolerance; processor reliability; hyperscale computing; hardware faults; low-precision computing; AI training; radiation-induced faults; distributed systems.

I. INTRODUCTION

For most of computing history, correctness at the hardware level was treated as a given. A processor was expected to either produce the right answer or fail in a way the system could observe, a crash, a parity error, a machine check exception. Software engineers built entire disciplines, distributed consensus, replication, checkpointing, on top of that assumption, worrying about network partitions and disk failures but rarely about the arithmetic unit itself lying to them.

That assumption is breaking down. Between 2021 and the present, engineering teams at Meta and Google independently published accounts of production CPUs that intermittently compute wrong results with no accompanying signal, what Google researchers called mercurial cores and what the broader literature now calls silent data corruptions, or SDCs (Dixit et al., 2021; Hochschild et al., 2021). Both teams found the rate of these events to be orders of magnitude higher than the one in a million occurrence predicted by chip makers' own fault injection testing. Follow up work at Alibaba scale confirmed the phenomenon across a different fleet and vendor mix (Wang et al., 2023). The problem is not a defect confined to one company's hardware supply chain; it is a systemic property of how modern silicon is built and operated.

This paper makes three claims. First, the structural drivers of SDCs, transistor density, reduced voltage margins for power efficiency and the sheer scale of modern deployments, are not going away; if anything they compound over the next decade as chips are pushed harder for AI workloads. Second, hardware alone cannot close this gap: full redundancy is too costly for commodity fleets and the long tail of possible fault sites defeats targeted screening. Third, a promising, underused answer already exists in the algorithms literature: algorithm based fault tolerance (ABFT) and its descendants, which detect and often correct errors using the mathematical structure of the computation itself, at a fraction of the cost of full duplication. The paper proposes this area, specifically its application to AI

training and quantized, radiation exposed computing, as a strong and timely thesis topic for a computer science student.

II. BACKGROUND: THE END OF THE FREE RELIABILITY ASSUMPTION

Why the hardware is becoming less trustworthy

Three converging trends explain why SDCs are increasing rather than shrinking. The first is transistor density. As feature sizes shrink, the electrical charge that represents a stored bit becomes smaller and the margin between a valid one and a valid zero narrows, making individual transistors more susceptible to marginal timing and voltage conditions that used to be harmless. The second is aggressive voltage and frequency scaling, used industry wide to control power draw and heat in dense datacenter racks; operating closer to the edge of a chip's validated envelope improves efficiency but shrinks the safety margin that once absorbed manufacturing variation and aging effects. The third is scale itself. A failure mode that occurs once in a million operations is irrelevant on a single workstation and unavoidable across a fleet of hundreds of thousands of machines running trillions of operations a day. Dixit et al. (2021) note explicitly that increased density, technology scaling and wider datapaths raise the probability of silent errors and that the blocks most exposed are often those without dedicated error correction, unlike DRAM or SRAM, which typically carry ECC protection.

Evidence from hyperscale fleets

The Meta study describes a real production incident in which a corrupted core silently altered values inside a file decompression pipeline, producing incorrect application output with no crash and no error log entry, discovered only because downstream results looked wrong (Dixit et al., 2021). The Google study, appearing at HotOS 2021, coined the term mercurial cores for CPUs that intermittently and reproducibly miscompute specific instructions under specific operating conditions and reported an observed rate on the order of one defective core per thousand machines, roughly a thousand times higher than vendor fault injection estimates (Hochschild et al., 2021). Both papers stress the same uncomfortable conclusion: the failures are fail-silent, meaning the hardware itself gives no indication anything went wrong and detection requires either redundant computation or software level plausibility checks built into the application.

A subsequent large scale study of a production CPU population, conducted at Alibaba and later published through SOSP and ACM Transactions on Computer Systems, corroborated the pattern across a different hardware base, reinforcing that this is not a single vendor's manufacturing anomaly but a cross industry phenomenon tied to how modern chips are built and operated (Wang et al., 2023).

SDCs and AI training: a multiplier effect

Nowhere does the cost of silent corruption compound faster than in large scale AI training. A single foundation model training run can occupy tens of thousands of accelerators continuously for weeks and because each optimizer step depends on the state produced by the previous one, a single corrupted value, an exponent bit flipped inside a tensor core, a miscomputed gradient, does not stay local. It propagates forward through every subsequent step and in distributed training it can propagate outward across the whole cluster through gradient synchronization. Reporting from Meta's Llama 3 training run attributed roughly 1.4 percent of unexpected GPU interruptions to this class of fault (as cited in recent GPU SDC characterization work; see George et al., 2026 and related anatomy studies of GPU error patterns). Because training workloads are inherently somewhat noise tolerant, a wrong value does not always crash the job; it can instead quietly bias the model, which is arguably worse, since the corruption is invisible until the model's behavior is scrutinized long after the compute budget has been spent.

This is precisely the mechanism described informally as spin on a bowling ball: a small, repeated, uncorrected bias does not average out the way random noise does, because it is not random. It is systematic, coming from the same defective functional unit on the same core, striking the same code path again and again. Over enough steps, the trajectory of the model or the trajectory of any long running distributed computation, ends up meaningfully different from what it would have been on healthy hardware.

III. WHY HARDWARE ALONE CANNOT SOLVE THIS

The intuitive response to SDCs is to fix them at the hardware level: better manufacturing screening, more built in redundancy, error correcting logic on every functional unit. Three practical constraints make this insufficient on its own.

- Cost. Full hardware redundancy, such as triple modular redundancy at the level of every arithmetic unit, roughly triples silicon area and power for that unit. At datacenter scale this is economically incompatible with commodity computing, where margins are already thin and density drives cost efficiency.
- The long tail of fault sites. A modern CPU or GPU contains an enormous number of distinct functional blocks and Hochschild et al. (2021) note that most of these are not amenable to error correcting codes the way memory and network links are, because coding techniques work best when errors can be checked against a large, self contained chunk of data; a single miscomputed instruction inside an arithmetic pipeline does not offer that structure cheaply.
- Detection versus correction cost asymmetry. Even where hardware detection is feasible, GPU SDC characterization work notes that detection alone typically requires roughly double the computation (dual modular redundancy with comparison), while correction requires roughly triple the computation (triple modular redundancy with voting), a cost multiplier that is very hard to justify for every operation in a training run that already consumes enormous energy and capital (see GPU error pattern studies referenced in Section 5.3).

Because full hardware correction does not scale economically and because vendors are reluctant and in the published literature largely unable, to name the specific defect mechanisms behind individual mercurial cores, the practical response in industry today is coarse: detect a misbehaving node through downstream symptoms, evict it from the fleet and roll back the affected work. This is expensive, reactive and crucially, does nothing to catch corruption that has already silently contaminated a result before eviction happens. It motivates a complementary, software level layer of defense.

IV. THE SPACE DIMENSION: RADIATION, QUANTIZATION, AND COMPOUNDING RISK

The reliability problem described above intensifies sharply once computing moves off the surface of the Earth. Spacecraft electronics are continuously exposed to cosmic rays, solar particle events and trapped radiation belts, which induce Single Event Effects (SEEs) in semiconductor devices. The dominant failure mode is the Single Event Upset (SEU), a transient bit flip in memory, a register or a logic gate caused by a single energetic particle strike, alongside rarer but more damaging Single Event Latchups (SELs), which can physically overheat and destroy a device if not caught quickly (Mehlitz and Penix, NASA Ames; Wang et al., ASPLOS 2025).

Historically, the response was hardware centric: use radiation hardened (rad hard) processors, fabricated with specialized processes and design techniques specifically to resist SEEs. But rad hard parts lag several process generations behind commercial silicon, are far more expensive, consume more power and mass budget and are increasingly seen as a poor fit for the computational demands of modern autonomous spacecraft, onboard machine learning and dense sensor processing. This has driven a well documented shift toward commercial off the shelf (COTS) processors in space missions, including NASA's Ingenuity Mars helicopter, which ran Linux on an unmodified commercial Snapdragon system on chip (Mehlitz and Penix; Wang et al., ASPLOS 2025). COTS hardware is fast, cheap and power efficient, but it is not radiation hardened, which means the SEU rate a mission must tolerate goes up substantially compared to a traditional rad hard architecture.

This is the same structural tension described in Sections 2 and 3, cost and scale pushing systems toward less reliable hardware, except radiation adds an external, uncontrollable error source on top of the terrestrial causes already discussed. And the consequences of an uncaught error are magnified by two further factors the user's framing correctly identifies. First, quantization: as onboard AI inference and control systems move to lower precision number formats to save power and memory bandwidth, each remaining bit carries more information, meaning a single flipped bit has a proportionally larger effect on the represented value than it would in a wider, higher precision format. Second, irrecoverability: a spacecraft cannot be walked over to and rebooted by a technician, so an SEU that goes undetected until it has corrupted control logic or navigation state can be mission ending rather than merely inconvenient.

Existing space grade mitigations mirror the ABFT philosophy at a coarser grain: triple modular redundancy across whole processors with a voting circuit, periodic memory scrubbing to purge latent bit flips before they accumulate and increasingly, purely software based schemes that monitor COTS hardware behavior and predictively reboot or checkpoint before an SEL

causes permanent damage (Czajkowski and Strobel; Wang et al., ASPLOS 2025). These are promising precedents for exactly the kind of lightweight, software first fault tolerance this paper argues will be needed more broadly, but most existing space software mitigations operate at a coarse level, whole process checkpointing, memory scrubbing, rather than inside the arithmetic of the algorithm itself. That gap, applying fine grained, algorithm aware fault tolerance to onboard, quantized computation, is a natural extension of the terrestrial ABFT literature into the space domain.

V. ALGORITHMIC FAULT TOLERANCE: FOUNDATIONS AND STATE OF THE ART

Foundations: checksum based ABFT

Algorithm based fault tolerance (ABFT) predates the current SDC crisis by nearly four decades. The foundational idea, introduced by Huang and Abraham for matrix operations, is to augment a matrix with extra checksum rows and columns computed as linear combinations of the original data, then carry those checksums through the computation alongside the real data. Because operations like matrix multiplication are linear, the checksum relationship that held for the inputs should still hold for the output; if it does not, an error occurred and in many cases its location can be pinpointed and corrected from the pattern of the checksum mismatch. The elegance of the approach is that the extra computation is proportional only to the size of the checksum vectors, not to the size of the full matrix operation, so the overhead is a small fraction of the total work rather than a multiple of it.

This is fundamentally different from replication based fault tolerance. Where triple modular redundancy triples the work to get one vote among three answers, ABFT gets error detection and often correction, essentially for free by exploiting the mathematical structure that the algorithm already has. Subsequent work extended the original row and column checksum scheme to block checksums suited to large scale parallel systems, diagonal checksums capable of correcting two simultaneous errors rather than one and adaptations for sparse matrices, LU decomposition, Cholesky and QR factorization and other core linear algebra kernels widely used in scientific and machine learning workloads (see the survey literature summarized in Bosilca et al. and Du et al., as referenced in later ABFT papers).

Modern extensions: coded and verifiable computing

Recent work has generalized the checksum idea into what is often called coded computing, treating a distributed computation the way a communications engineer treats a noisy channel, adding redundancy in a structured way so that errors can be detected or corrected without resending, or in this case, recomputing, the entire job. Block checksum schemes designed for exascale style parallel systems reduce the communication overhead that plagued naive row and column checksums when data is spread across thousands of nodes (see block checksum literature for large scale parallel matrix multiplication). More recent proposals push ABFT concepts into adversarial and verifiable computing settings, for instance combining lightweight checksum style verification with encrypted computation to catch not just accidental bit flips but deliberate tampering, at overhead far below cryptographic proof systems like zero knowledge proofs.

GPU and AI specific fault tolerance

The AI training case requires adapting ABFT to modern accelerator hardware, tensor cores, mixed precision arithmetic and enormous single operation sizes, where classical row and column checksum schemes were not originally designed to run. Recent studies have characterized how SDCs actually manifest inside GPU tensor cores and CUDA cores at the level of individual bit flips, finding that the corruption signature depends heavily on numeric format: a flipped exponent bit in a narrow format like FP16 can shift a value by orders of magnitude, while the same fault in a wider format like BF16 produces a smaller deviation and aggressively compressed formats like FP8 change the tradeoff again (recent GPU SDC anatomy and LLM PRISM studies, 2026).

Building directly on classical ABFT, newer proposals fuse lightweight checksum verification into the GPU kernel itself so it runs immediately after an otherwise unmodified tensor core matrix multiply, rather than as a separate, expensive full recomputation; one such approach uses a compact sum sketch to flag corruption on every call and reports overhead far below full duplication, though like classical row and column checksums it can typically localize only a single error per check (post hoc GPU matrix multiplication fault recovery literature, 2026). Related work applies syndrome decoding, a technique borrowed from error correcting codes, to quantized integer GPU arithmetic specifically, aiming for exact, per element localization of a corrupted value so that a degrading accelerator

can be identified and quarantined rather than merely flagged after the fact.

The overhead to coverage tradeoff

Every algorithmic fault tolerance scheme sits on a spectrum between cost and coverage. A pure checksum check might add a small, roughly single digit percentage of extra computation and catch a single corrupted element reliably, but fail silently itself if two errors happen to cancel out in the checksum. Full duplication catches essentially everything but doubles or triples cost outright. Google and Meta's own operational experience puts a number on this asymmetry: production grade detection through duplication and comparison costs roughly double the baseline computation and correction through voting costs roughly triple, which is exactly the kind of overhead that motivates lighter weight, algorithm aware alternatives (see Section 3 and the GPU SDC anatomy literature).

This tradeoff is the crux of the argument in the user's framing: accepting a modest, bounded performance loss, on the order of a few percent, can be worth it if it meaningfully raises the odds of catching a mercurial core before its errors propagate silently across a cluster or across the steps of a training run. Quantifying exactly where that modest cost buys the most protection and for which classes of workload, is an open and tractable research question and it is the core of the thesis direction this paper proposes.

VI. PROPOSED THESIS DIRECTION

Research questions

- How does the detection coverage of lightweight, checksum based ABFT schemes degrade as numeric precision drops from FP32 to BF16, FP16, FP8 and quantized integer formats commonly used in AI training and inference and where does coverage fall off a cliff versus degrade gracefully?
- What is the minimum overhead, measured as a percentage of added computation or wall clock time, required to detect a single silently corrupted value with a target confidence level, for representative kernels used in large scale model training (matrix multiplication, attention, gradient reduction)?
- Can algorithm aware checksums be layered with the coarse, node level eviction strategies already used in production (Section 3) to catch corruption earlier in its propagation, reducing the amount of wasted compute before a defective accelerator is identified?
- Do the same lightweight, checksum based techniques transfer usefully to radiation induced, quantized onboard computing in space, where error sources are external (cosmic rays) rather than internal (manufacturing variation) and where recovery options are far more limited?

Proposed methodology

A workable thesis project could combine controlled fault injection with real workload measurement in three phases. First, implement or extend an existing GPU fault injection framework, such as the tools described in recent GPU SDC characterization literature, to inject representative bit flip patterns into tensor core and CUDA core operations across several numeric formats. Second, implement one or more ABFT style checksum schemes at the kernel level for a representative training workload, such as a small transformer and measure both detection coverage and added wall clock overhead as precision and matrix size vary. Third, extend the same checksum logic to a quantized integer inference workload representative of onboard spacecraft or edge computing and evaluate it under simulated single event upset patterns drawn from published space radiation fault models, rather than requiring access to an actual beam testing facility.

Expected contribution

The expected contribution is an empirical map of the overhead to coverage curve for algorithmic fault tolerance across the precision levels most relevant to modern AI systems, paired with a concrete demonstration that the same lightweight techniques generalize from hyperscale datacenter training to radiation constrained, quantized space computing. That combination, terrestrial AI reliability and space reliability treated as two instances of the same underlying algorithmic problem, is not yet well represented in the published literature and would make for a distinctive, timely and practically grounded thesis.

VII. CONCLUSION

The evidence from Meta, Google and subsequent large scale studies is unambiguous: modern processors, tested and certified as they are, occasionally compute wrong answers with no signal that anything went wrong and they do so at rates far higher than the industry's own fault models predicted. The

forces behind this, transistor density, voltage scaling for efficiency and deployment scale, are not temporary manufacturing hiccups; they are the direct consequence of how the industry continues to push performance and efficiency and they will intensify as AI training pushes toward ever larger, ever more tightly synchronized clusters running ever lower precision arithmetic and as computing extends further into radiation environments that add an entirely separate, external source of the same underlying problem.

Hardware alone cannot absorb this cost economically. The path forward, already visible in decades old algorithm based fault tolerance research and its modern GPU and coded computing descendants, is software that assumes the hardware beneath it is occasionally, quietly wrong and that pays a small, well understood and bounded performance cost to catch that wrongness before it spreads. That is not a pessimistic conclusion about computing. It is a return to an older, humbler engineering habit: build systems that stay correct even when their parts are not perfectly reliable, rather than systems that simply assume they always will be.

REFERENCES

1. Bosilca, G., Delmas, R., Dongarra, J., & Langou, J. Algorithm Based Fault Tolerance Applied to High Performance Computing. https://arxiv.org/pdf/0806.3121
2. Czajkowski, D. R., & Strobel, D. J. High Performance, Radiation Hardened, Ultra Low Power Space Computer Leveraging COTS Microelectronics with SEE Mitigation. Space Micro Inc. https://klabs.org/mapld05/abstracts/138_czajkowski_1_a.html
3. Dixit, H. D., Pendharkar, S., Beadon, M., Mason, C., Chakravarthy, T., Muthiah, B., & Sankar, S. (2021). Silent Data Corruptions at Scale. arXiv:2102.11245. https://arxiv.org/abs/2102.11245
4. George, N., Gurumurthi, S., Sridharan, V., Dixit, H. D., Goksu, E., Parthasarathy, B., Huffman, A., Macieira, T., Sinha, A., Liberty, D., Minwell, L., & Chappell, R. S. (2026). Silent Data Corruption in Artificial Intelligence: A Growing Challenge for Large-Scale Machine Learning. IEEE Micro, 46.
5. Gizopoulos, D., Papadimitriou, G., Chatzopoulos, O., Karystinos, N., Dixit, H. D., & Sankar, S. (2024). Silent Data Corruptions in Computing Systems: Early Predictions and Large-Scale Measurements. IEEE European Test Symposium (ETS).
6. Hochschild, P. H., Turner, P., Mogul, J. C., Govindaraju, R., Ranganathan, P., Culler, D. E., & Vahdat, A. (2021). Cores that don't count. Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS '21). https://doi.org/10.1145/3458336.3465297
7. Mehlitz, P. C., & Penix, J. Expecting the Unexpected: Radiation Hardened Software. NASA Ames Research Center / CSC. https://ntrs.nasa.gov/api/citations/20060015096/downloads/20060015096.pdf
8. Saxena, N., Hukerikar, S., & Hari, S. (2026). Silent Data Corruption: Optimal Mitigation Strategies for Data Center Computing. IEEE Micro, 46(1), 10-18.
9. Wang, H., Myint, S., Verma, V., Winetraub, Y., Yang, J., & Cidon, A. (2025). Radshield: Software Radiation Protection for Commodity Hardware in Space. ASPLOS '25.
10. Wang, S., Zhang, G., Wei, J., Wang, Y., Wu, J., & Luo, Q. (2023/2024). Understanding Silent Data Corruptions in a Large Production CPU Population. Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP); also ACM Transactions on Computer Systems (TOCS), 2024. https://doi.org/10.1145/3600006.3613149
11. GPU Error Pattern Study and Modeling Guidance: The Anatomy of Silent Data Corruption. arXiv:2605.04213. https://arxiv.org/html/2605.04213v1
12. LLM-PRISM: Characterizing Silent Data Corruption from Permanent GPU Faults in LLM Training. arXiv:2604.10390. https://arxiv.org/html/2604.10390v1
13. Sketching the Error, Not the Product: Post Hoc Fault Recovery for Half Precision GPU Matrix Multiplication (FP-Sketch). arXiv:2609.19758. https://arxiv.org/html/2609.19758
14. Syndrome Decoding for Silent Data Corruption in Quantized Integer GPU Arithmetic. arXiv:2609.19743. https://arxiv.org/html/2609.19743