CROSS-SCALE MODEL RESILIENCE FOR LLM INFERENCE UNDER HARDWARE FAULTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern large-language models (LLMs) deployed on computing systems at scale are increasingly exposed to transient faults. These faults can silently corrupt persistent model state (i.e., parameters), affecting subsequent inference and potentially causing consequential decisions and actions. Existing resilience mechanisms detect and correct corruptions in bits, activations, and parameters. However, bounding the numerical errors from such corruptions does not guarantee semantically correct outputs. In this work, we present cross-scale model resilience (CSMR), a mechanism that detects and mitigates fault-induced errors at the output level. CSMR pairs a target model with a lightweight auxiliary model and measures their token- and sequence-level semantic discrepancy, triggering correction only when the discrepancy exceeds a calibrated threshold. To minimize overhead, we integrate CSMR into speculative decoding, using the auxiliary as the drafter and embedding discrepancy checking into the acceptance stage. Upon detection, CSMR selectively reloads corrupted parameters. Across multiple LLMs and benchmarks under diverse fault models, CSMR restores accuracy from 0% to 95.6–99.6%, with less than 0.4% latency overhead in fault-free executions.

1 INTRODUCTION

LLM inference is becoming a dominant workload in modern high-performance and distributed systems (Zhang et al., 2025; Nazi & Peng, 2024; Lai et al., 2024; Wang et al., 2025; Xu et al., 2025a). As deployments scale to billions of tokens across increasingly dense and heterogeneous environments, exposure to transient hardware faults (soft errors) arising from sources such as cosmic radiation, shrinking transistors, and low-voltage operations (Nie et al., 2016; Fratin et al., 2018; Chandramoorthy et al., 2019; Kim et al., 2014)—becomes unavoidable (Zhu et al., 2025; Oles et al., 2024; Das et al., 2025; Coalson et al., 2025). When such faults corrupt model parameters, they can silently affect autoregressive decoding and alter generations without triggering fail-stop failures, resulting in silent data corruptions (SDCs) (Agarwal et al., 2023; Sun et al., 2025a; Dixit et al., 2021). Such SDCs pose a critical reliability challenge for large-scale LLM inference.

Protection mechanisms against such faults span hardware- and software-level techniques (Tahmasivand et al., 2025; Guo et al., 2021; Asgari Khoshouyeh et al., 2023; Schmedding et al., 2024; Chen et al., 2021; Sun et al., 2025b). However, existing approaches share a common assumption: they ensure inference correctness by preserving the *numerical integrity* of internal model states, which is poorly aligned with the error characteristics of LLM inference. Numerical perturbations may leave the generated output unchanged, whereas even small deviations can induce discrete decoding divergence and semantically incorrect outputs (Sun et al., 2025a). This mismatch can cause protection mechanisms to (a) over-trigger on benign perturbations or (b) miss semantically critical faults that remain within expected numerical ranges.

Contributions. In this work, we introduce a shift from numerical correctness to semantic-level correctness for LLM inference. We argue that fault detection should operate on observable generation behavior rather than internal numerical states. Resilience does *not* rely on comprehensively protecting billions of parameters and intermediate activations. Instead, protection is applied *only when* perturbations manifest as semantically meaningful divergence. This enables scalable, task-agnostic resilience, avoiding unnecessary corrections for perturbations that do not affect output behavior.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We instantiate this paradigm through Cross-Scale Model Resilience (CSMR), a decoding-level resilience enhancement for autoregressive LLM inference. Our *key insight* is that the model itself provides a natural reference for detecting abnormal behavior: discrepancies between models of different scales reveal semantically meaningful deviations. CSMR pairs a large target model with a lightweight auxiliary model and performs cross-scale discrepancy checking during decoding. Recovery is triggered only when output-level divergence exceeds a threshold.

CSMR addresses three key challenges: (1) To reliably distinguish harmful from output-preserving faults, we introduce a normalized discrepancy metric that captures two complementary signatures of semantic divergence—local probability spikes and distributional drift—that standard divergence measures may miss. (2) To efficiently localize corrupted parameters among billions of weights, we use a hash-based fault isolation mechanism that identifies affected weight tensors without full model inspection. (3) To integrate CSMR into latency-sensitive inference pipeline, we embed discrepancy checking into speculative (draft–verify) decoding (Leviathan et al., 2023) and reuse the auxiliary model as the draft generator, amortizing resilience overhead within standard inference.

We conduct a comprehensive evaluation across three LLM families (8–32B) and three benchmarks. Under our primary single-weight 2-bit fault setting, CSMR restores accuracy to 96.9–99.4%; this improved resilience comes with little runtime overhead: CSMR adds only 0.1–0.4% latency under fault-free and benign conditions. In contrast, methods that comprehensively check model weights add 4.1–31.0% under benign faults by repairing corruptions that leave answers unchanged. CSMR also generalizes to two-weight 2-bit and single-weight 3-bit faults, achieving 95.6–99.6% accuracy under critical faults. Even when both the target and draft are corrupted, it detects 93.4–98.1% of critical faults in our joint-fault experiments. Overall, our results demonstrate that numerical fault detection alone is neither necessary nor sufficient for efficient, fault-resilient LLM inference.

2 Background and Related Work

LLM inference. Most LLMs adopt decoder-only *transformer architectures* (Vaswani et al., 2017; Radford et al., 2018) and generate tokens autoregressively. Each transformer block consists of self-attention and multi-layer perceptron (MLP) modules, with linear projections forming the core parameterized operations and the query, key, and value projections in self-attention. LLM inference proceeds in two phases. During *prefilling*, the model processes the input prompt and constructs the key–value (KV) cache that stores attention states for previous tokens. During *decoding*, it generates output tokens sequentially by attending to cached KV states without recomputing prior tokens.

Transient hardware fault in LLM inference. Due to the scale and computational demands, LLMs are deployed on systems equipped with large-memory accelerators, increasing their exposure to transient hardware faults (i.e., *soft errors*) (Tiwari et al., 2015; Oles et al., 2024; Zhu et al., 2025). Such faults can be caused by cosmic radiation, shrinking transistors, and low-voltage operations (Fratin et al., 2018; Nie et al., 2016; Chandramoorthy et al., 2019; Kim et al., 2014), corrupting model weights or intermediate activations during inference. These errors may not be explicitly detected during execution, leading to SDCs in generated outputs (Agarwal et al., 2023; Sun et al., 2025a).

Memory faults corrupt model weights and can repeatedly affect downstream computations as the corrupted weights are reused. In contrast, *computational faults* affect individual operations, exhibit more localized propagation, and are often attenuated by subsequent nonlinearities and normalization layers (Li et al., 2017; Sun et al., 2025a). We primarily focus on memory faults because persistent weight corruptions can influence multiple decoding steps and induce semantic divergence in outputs. Prior work has shown that even a small number of bit errors in model weights can drastically alter model behavior (Coalson et al., 2025; Rakin et al., 2019; Das et al., 2025; Xu et al., 2025b). We also evaluate against computational faults in Appendix E. We model memory faults as *multi-bit flips* within a weight value. This choice reflects a realistic fault space in modern accelerators: single-bit errors are corrected transparently by on-die ECC (e.g., NVIDIA Volta and Ampere GPUs (NVIDIA, 2017; 2020)), while multi-bit errors remain uncorrected (Oles et al., 2024; Zhu et al., 2025).

Fault protection mechanisms. (1) *Hardware-level protection mechanisms* such as ECC (Hamming, 1950; Hsiao, 1970; Slayman, 2005) are widely adopted in modern GPU memory systems, providing single-error correction and double-error detection (SEC–DED) capabilities. However, multi-bit upsets and intermittent defects can evade ECC protection (Oles et al., 2024; Zhu et al., 2025). (2)

System-level mechanisms protect model states or computations at runtime. Redundancy-based techniques replicate computations or model components to mask faults (Milluzzi & George, 2017; Yang et al., 2021; Asgari Khoshouyeh et al., 2023; Schmedding et al., 2024). DeepDyve (Li et al., 2020) uses a compressed checker model to verify classifier outputs and triggers recomputation upon disagreement, but its discrete-label comparison does not directly extend to autoregressive generation. (3) *Algorithm-based fault tolerance (ABFT)* encodes checksums into computations and verifies them at runtime (Liu et al., 2024; Xue et al., 2023). Recent LLM extensions (Liang et al., 2025) target only transient arithmetic faults rather than persistent memory corruptions. (4) *Hash-based integrity verification* detects weight corruption by comparing runtime signatures with precomputed references. HASHTAG (Javaheripi & Koushanfar, 2021; Javaheripi et al., 2022) and ModelShield (Guo et al., 2021) verify weight tensors, while LM-Fix (Tahmasivand et al., 2025) detects bit-flips via hashes of layer outputs on fixed inputs. Such methods detect weight corruption regardless of whether it affects the generated output. (5) *Model-level protections* leverage training or architectural constraints. Hessian-aware training (Prato et al., 2025) reduces parameter sensitivity through curvature-aware regularization, while Aegis (Wang et al., 2023) uses a multi-exit architecture with randomized internal classifiers. Both methods require changes to training or model architecture, limiting their applicability to LLMs. (6) *Range-restriction methods* constrain activations within bounds obtained through offline profiling (Chen et al., 2021; Sha et al., 2024; Zheng et al., 2025; Sun et al., 2025b). But faults that remain within these numerical bounds can still alter decoding decisions and outputs.

3 CSMR: Cross-Scale Model Resilience

3.1 The gap between numerical errors and semantic divergence

In LLM inference, error propagation differs from classical numerical computation. We show that it is *sparse, input-dependent, and discrete*, such that numerical deviation is an unreliable proxy for output-level correctness.

Discrete branching amplifies perturbations selectively. At each decoding step t, the model selects a token via $y_t = \arg\max_{v \in \mathcal{V}} P_\theta(v|y_{<t})$, where P_θ denotes the output distribution conditioned on the prefix $y_{<t}$. A weight perturbation δ to model parameters θ modifies the logits and the resulting distribution. But as long as the ordering of candidate tokens remains unchanged,

$$\arg\max_{v \in \mathcal{V}} P_\theta(v|y_{<t}) = \arg\max_{v \in \mathcal{V}} P_{\theta+\delta}(v|y_{<t}),$$

the generated token is unaffected and the perturbation is fully *masked*. In contrast, when a perturbation crosses a decision boundary and alters the top-ranked token, even by a small margin, the selected token changes ($y_t \neq y'_t$). This corrupted token becomes the prefix for subsequent steps, causing the decoding trajectory to branch out *irreversibly*. Autoregressive dependency then amplifies the deviation. Thus, a single token change at step t can cascade into entire output corruption.

Numerical correctness does not equate to output correctness: (1) A large perturbation that shifts logits without changing the argmax has *zero* observable impact, while (2) a small perturbation that flips the argmax decision produces *complete* semantic divergence. The relationship is therefore highly nonlinear and sparse: the vast majority of faults are benign and leave the generated sequence intact, while a small fraction triggers disproportionate damage. Figure 1 illustrates this bimodal distribution: under random 2-bit fault injection across diverse inputs, output divergence is either negligible or near-total, with little in between. This decoupling exposes both classes of existing resilience mechanisms to structural failures. Methods that correct all detected errors (Asgari Khoshouyeh et al., 2023; Tahmasivand et al., 2025; Guo et al., 2021) pay the full cost of recovery, even though the vast majority are benign, wasting effort on (1). Activation-bounding methods (Chen et al., 2021; Sun et al., 2025b; Sha et al., 2024; Roquet et al., 2024; Ma et al., 2024) constrain numerical magnitudes but cannot necessarily

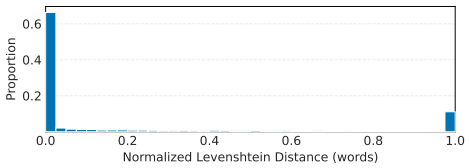

Figure 1: **Distribution of normalized word-level Levenshtein distance between fault-free and fault-injected generations** across 4,500 memory faults (using our setup from §4.2). Fault impact on LLMs is bimodal: faults either leave the generation unchanged or corrupt it almost entirely.

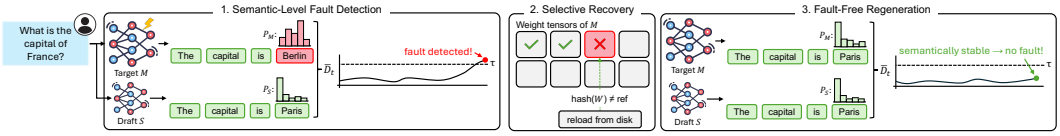

Figure 2: **Overview of CSMR's workflow.** We detect faults via cross-scale output divergence, selectively reload corrupted weight tensors, and regenerate model outputs.

detect (2), since argmax-flipping perturbations need not produce out-of-bound activations. Critically, neither family operates on the signal determining correctness: the model's decoding behavior.

These observations lead to the following design requirements for resilient LLM inference:

- **R1: Output-aware detection.** Because output correctness is determined by discrete token selection, fault detection must operate at the decoding phase. Mitigation mechanisms should identify abnormal behavior through output-level divergence signals, not internal numerical metrics.
- **R2: Selective recovery.** Given the sparse and input-dependent nature of fault impact, effective mechanisms invoke correction *only* for corruption that meaningfully affects generated outputs.
- **R3: Deployment compatibility.** Mechanisms should integrate seamlessly into existing LLM inference pipelines without requiring retraining, full model replication, or specialized hardware.
- **R4: Low amortized overhead.** Because critical faults are rare, mechanisms that activate only upon detected divergence can amortize their cost across predominantly fault-free execution.

3.2 CSMR OVERVIEW

As shown in Figure 2, CSMR pairs a large target model M with a smaller draft model S, both of which produce token-level probability distributions over the same decoding context (1). CSMR computes a discrepancy metric and evaluates it against a calibrated threshold τ. If divergence exceeds τ, it isolates and recovers corrupted weights (2) and restarts generation (3). Our design realizes the requirements from §3.1: **R1** through output-aware discrepancy detection and offline calibration, **R2** through threshold-gated selective recovery, **R3** through deployment without retraining or full-model replication, and **R4** through selective weight recovery and speculative decoding integration, which reduce recovery cost and amortize the draft-model overhead, respectively.

3.3 CROSS-SCALE DISCREPANCY DETECTION

A fault may concentrate probability mass on one outlier token or redistribute it diffusely across many. We therefore combine two complementary scores in a per-token discrepancy metric.

Our metric. Given input p and the current decoding prefix c generated by M, both models produce next-token distributions over their shared vocabulary $\mathcal{V}$:

$$P_M = M(\cdot \mid p, c), \quad P_S = S(\cdot \mid p, c).$$

To capture local discrepancies, we use the *negative log-likelihood (NLL)* of M's top token under P_S:

$$D_{\text{local}} = -\log P_S(t_M), \tag{1}$$

where $t_M = \arg\max_{v \in \mathcal{V}} P_M(v)$ denotes the top-ranked token from M. This term becomes large when M sharply promotes a token that is unlikely under S, making it sensitive to localized spikes. For global discrepancy, we compute the *JS divergence* (Lin, 1991) between P_M and P_S:

$$D_{\text{global}} = \frac{1}{2}\left(\text{KL}(P_M \,\|\, \bar{P}) + \text{KL}(P_S \,\|\, \bar{P})\right), \tag{2}$$

where $\bar{P} = \frac{1}{2}(P_M + P_S)$ is their mean mixture and $\text{KL}(\cdot||\cdot)$ is the KL divergence (Kullback & Leibler, 1951). Unlike NLL, this term becomes large when probability mass shifts are distributed across multiple tokens, making it sensitive to broader changes in the overall output distribution.

To jointly detect local and global discrepancies, we treat a prediction as divergent when either signal exhibits abnormal deviation. We therefore take the maximum of both scores:

$$D = \max(\hat{D}_{\text{local}}, \hat{D}_{\text{global}}), \tag{3}$$

where $\hat{D}_{\text{local}}$ and $\hat{D}_{\text{global}}$ are normalized versions of D_{local} and D_{global} to place them on a comparable scale; our normalization procedure is described in §3.4.

Windowed fault detection. As M and S differ in scale and parameterization, modest output disagreement naturally occurs even during fault-free decoding (Leviathan et al., 2023; Chen et al., 2023). Thus, thresholding the per-token discrepancy score D would produce excessive false positives, since isolated token-level deviations are common and often benign. We therefore aggregate discrepancy scores over a sliding window and evaluate their average behavior. Specifically, for window size w, we define the *aggregated discrepancy metric* at decoding step t as:

$$\bar{D}_t = \frac{1}{w} \sum_{i=t-w+1}^{t} D_i, \tag{4}$$

where D_i denotes the per-token discrepancy score at token position i. We trigger recovery only when $\bar{D}_t > \tau$, ensuring that isolated disagreements do not cause unnecessary recovery. We set w to 10, which best balances detection effectiveness with false-positive suppression (see Appendix G).

3.4 Offline calibration

Setting an appropriate stability threshold τ is critical for distinguishing benign model-level disagreement from fault-induced semantic breakdown. To normalize the two discrepancy scores and set τ without fault-specific profiling, we calibrate once per target–draft pair using fault-free generations.

Offline data collection. We first estimate the benign distributions of our discrepancy metrics—D_{local} and D_{global}—using general-purpose chat data. Specifically, we sample 5,000 instructions from UltraChat 200k (Ding et al., 2023), a large curated collection of ChatGPT-generated instructions, and denote the resulting calibration set as $\mathcal{D}_{\text{cal}} = \{p_i\}_{i=1}^{5000}$. For each $p_i \in \mathcal{D}_{\text{cal}}$, we greedily generate a completion c_i of up to 1,024 tokens using M. Because this stage is performed offline, we assume fault-free generation, consistent with prior work (Chen et al., 2021; Sun et al., 2025b). Finally, we compute D_{local} and D_{global} for every token in each completion c_i, yielding empirical benign distributions for both metrics that serve as the basis for subsequent threshold calibration.

Metric normalization. Because D_{local} and D_{global} operate on different scales—NLL is bounded in $[0, \infty)$, whereas JS divergence is bounded in $[0, \ln 2]$—we normalize both before applying $\max$ in Eq. 3. Specifically, as we find that the distributions of both metrics are heavily right-skewed under benign conditions, we apply robust normalization (scikit-learn developers):

$$\hat{D}_{\text{local}} = \frac{D_{\text{local}} - \text{med}_{\text{local}}}{\text{IQR}_{\text{local}}}, \; \hat{D}_{\text{global}} = \frac{D_{\text{global}} - \text{med}_{\text{global}}}{\text{IQR}_{\text{global}}},$$

where med and IQR denote the median and IQR, respectively, of each metric over all tokens in completions from $\mathcal{D}_{\text{cal}}$, thereby placing both normalized scores on a meaningfully comparable scale.

Threshold calibration. We calibrate τ from the distribution of per-sequence detection statistics over $\mathcal{D}_{\text{cal}}$. For each calibration sequence, the detection statistic is $\max_t \bar{D}_t$, i.e., the peak windowed aggregate of D across all tokens. We set τ to the 99.5-th percentile of these per-sequence statistics, which ensures a low false-positive rate while maintaining high detection rates (see Appendix G).

3.5 Selective recovery policy

After detecting a fault, CSMR rolls back the target model M. The dominant cost is recovery latency: as we show in Figure 3, naïvely reloading all weights of a 32B model takes up to 37.2 seconds on our hardware, which would turn every detection into a large latency spike (undermining **R4**).

Fault isolation and recovery. We use hashing to efficiently identify corrupted weight tensors. Prior to deployment, we compute a *reference hash* of each weight tensor in M. Because model weights reside in GPU memory and transferring them to the host for hashing would dominate recovery latency, we use a GPU-native hash function (SGLang Team) based on MurmurHash3 (Appleby) and SplitMix64 (Steele et al., 2014), implemented as a kernel that operates directly on GPU memory. The function produces a 64-bit digest, making the probability of a random collision negligible ($\approx 2^{-64}$). After CSMR detects a fault, we recompute each tensor's hash and selectively reload only tensors whose hashes have changed. After reloading, we re-hash the affected tensors to verify that the stored copies on disk are themselves intact; if a re-hashed tensor still mismatches its reference,

the stored copy is also corrupted, and CSMR flags the fault for manual recovery. Figure 3 shows that our selective reloading procedure reduces recovery time by 17.3–112.2× (0.1–0.3 seconds).

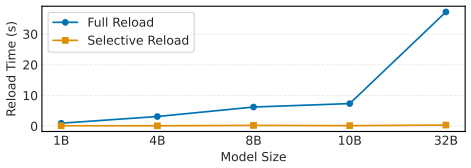

Figure 3: **Our selective weight recovery reduces overhead by 17.3–112.2×.** The recovery latency with and without our selective weight tensor reloading, averaged across 10 trials (95% CIs are within ±0.21 s across all data points). We report results for all models from Table 1 except Falcon3-1B-Instruct.

Regeneration after recovery. After selectively reloading any corrupted tensors, CSMR regenerates the response from scratch under the original prompt, since we cannot determine which token first affected the output. Our re-generation policy depends on the outcome of the hash-based isolation step. If one or more corrupted tensors were identified and reloaded, the root cause has been resolved, and regeneration proceeds normally under CSMR. If no corrupted tensor is found, the discrepancy was caused either by a transient computational fault or a false positive. In either case, the hash check has confirmed that all weight tensors are intact, so re-generation proceeds from a clean model state. However, to avoid recurring false positives, monitoring is disabled for this single regeneration and re-enabled afterward.

3.6 Integration with speculative decoding

Running inference with both a target and draft model doubles per-token compute in the worst case, violating **R4** if applied naïvely. Fortunately, this overhead can be amortized by integrating CSMR with speculative decoding (Leviathan et al., 2023; Chen et al., 2023), which already uses paired target–draft models. At a high level, given a prompt p and current decoding prefix c, speculative decoding first uses S to auto-regressively generate a *draft* completion of k tokens. The target model M then *verifies* the draft in one forward pass, computing its own output distribution at each drafted position and accepting a prefix of the draft that satisfies a verification criterion. CSMR is compatible with any draft–verify scheme that uses a smaller draft model *without modification*, since verification provides both P_M and P_S at all accepted token positions, exactly the distributions required to compute our metric. One exception arises when M accepts all k drafted tokens, where M generates an additional token with no corresponding distribution from S. However, this occurs only when M and S exhibit very high agreement, making it unlikely to coincide with fault-induced divergence.

4 Evaluation

4.1 Experimental setup

Models. We conduct our evaluation using three target–draft instruction-tuned LLM pairs (shown in Table 1) from the Llama3 (Grattafiori et al., 2024), Falcon3 (Team, 2024), and Qwen3 (Yang et al., 2025) families. We select target models from diverse families and parameter scales (8–32B) to assess generalization. Following standard speculative decoding practice (Chen et al., 2023; Leviathan et al., 2023), we pair each target model with a substantially smaller draft model from the same model family (8–10× smaller). This ensures that the draft models maintain sufficient agreement with the target model under benign conditions. All models are deployed in bfloat16 precision, the predominant format for LLM inference in practice (HuggingFace; Wang & Kanwar, 2019).

Target Model	Draft Model	Abbreviation
Meta-Llama-3-8B-Instruct	Llama-3.2-1B-Instruct	Llama3
Falcon3-10B-Instruct	Falcon3-1B-Instruct	Falcon3
Qwen3-32B	Qwen3-4B	Qwen3

Table 1: **Model pairs used in our evaluation.**

Benchmarks. We harness three benchmarks spanning diverse task domains: GSM8K (Cobbe et al., 2021), which consists of grade-school math problems; MMLU (Hendrycks et al., 2021b), which contains multiple-choice questions across subjects from elementary to professional levels; and APPS (Hendrycks et al., 2021a), which includes coding problems of varying difficulty.

Metrics. We evaluate CSMR using two primary metrics: *accuracy* and average end-to-end *latency* measuring resilience and runtime overhead, respectively. We also characterize detection effectiveness using true positive rate (TPR), the fraction of critical faults detected, and false positive rate (FPR), the fraction of detections under fault-free and benign fault injection, in §4.7 and Appendix G.

Baselines. We compare CSMR against four baselines from §2. From *range-restriction* methods, we select Ranger (Chen et al., 2021), which clamps activation outputs to bounds calibrated via offline profiling, and FT2 (Sun et al., 2025b), which improves resiliency by protecting LLM-specific layers. FT2 supports both online and offline bound estimation; we use its offline variant, as online bounds derived during inference would be corrupted by memory faults. For both methods, we profile bounds on our calibration data (§3.4) for a fair comparison. From *redundancy-based* methods, we select Structural Coding (Asgari Khoshouyeh et al., 2023), which augments weight matrices with extra rows to enable checksum-based detection and recovery. Finally, from *hash-based integrity verification*, we select LM-Fix (Tahmasivand et al., 2025), which detects weight corruption by comparing runtime hashes against reference tensors. We use official implementations for all baselines except Ranger, which lacks an LLM-compatible release; it is re-implemented based on FT2.

4.2 Fault injection methodology

Primary fault model. Unless otherwise stated, our memory-fault experiments flip two bits in a single weight (*single-weight 2-bit*), the smallest corruption beyond the correction capability of single-error-correcting ECC. In §4.6, we also evaluate two-weight 2-bit and single-weight 3-bit faults.

Fault injection location. In our main experiments, we inject faults into the target model M and assume the draft model S fault-free. This choice reflects a realistic deployment scenario. (1) Because S is 8–10$\times$ smaller than M, its smaller memory footprint reduces its exposure to memory faults. (2) Moreover, even if S is corrupted, the impact on system correctness is limited, as a corrupted draft can only degrade draft acceptance rates, not M's outputs. To validate its limited impact on system correctness, we evaluate CSMR under draft model faults in §4.7 and find it remains largely resilient.

Fault injection tool. We adopt the PyTorch-based fault-injection framework of Sun et al. (2025a), which intercepts and flips bits within weight tensors before generation. For each injection, we randomly select a weight tensor in the target model, choose a random weight within that tensor, and flip the specified number of randomly chosen bits. The fault persists throughout the generation.

Fault outcome stratification. Prior work (Sun et al., 2025a; Agarwal et al., 2023) typically evaluates resilience by sampling faults uniformly at random and reporting aggregate performance. But, as the majority of faults are benign (Appendix B), aggregate metrics are dominated by inconsequential cases, obscuring performance on the rare faults that compromise correctness. We therefore stratify faults by outcome (benign or critical) and report accuracy and latency for both classes independently.

Sampling and replay procedure. For each benchmark and target model, we first randomly sample up to *500 prompts* for which the model produces a correct answer under fault-free conditions; this sample size yields tight confidence intervals for accuracy (within $\pm$5.2%). All generations use greedy decoding and have a maximum of 1024 tokens for sufficient response length. Next, for each prompt, we sample one *benign fault* (that does not change the original answer) and one *critical fault* (that does) via the fault injection tool described above. Specifically, we iteratively inject random faults until we obtain one of each type, discarding intermediate attempts of the opposite class. Finally, for each method, we replay each fault (and the fault-free execution) by regenerating a response from the same prompt and record accuracy and latency. In rare cases, despite using a fixed seed, we observe unintended behavior (e.g., incorrect fault-free generations); we exclude such samples.

4.3 CSMR's effectiveness

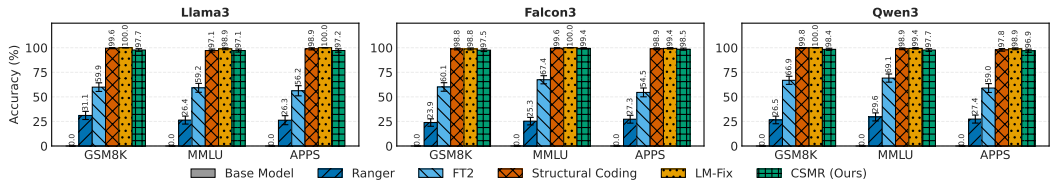

Figure 4: **CSMR recovers accuracy under single-weight 2-bit critical faults to 96.9–99.4%.** For each target–draft model pair, we report the *accuracy* of each mechanism. Our method achieves similar resilience to aggressive protection mechanisms while operating solely on output-level signals.

Figure 4 shows the accuracy of each baseline and CSMR under critical faults. Under the fault-free and benign-fault settings, accuracies remain uniformly high (96.9–100.0%) across methods.

CSMR significantly improves resilience to critical faults. Under critical faults, CSMR improves accuracy to 96.9–99.4%, demonstrating its effectiveness at preserving semantic correctness. In contrast, range-restriction mechanisms (Ranger and FT2) achieve only 23.9–69.1% accuracy—up to $\sim$4$\times$ worse. Thus, many critical faults do not induce activations correctable by these methods.

Structural Coding and LM-Fix achieve 97.1–100.0% accuracy. These methods offer marginal gains ($\sim$1%) over CSMR by protecting all model weights, but incur higher runtime cost in fault-free and benign settings (see §4.4 for analysis). CSMR achieves comparable protection while operating on output-level discrepancy signals, avoiding weight-level monitoring entirely. Moreover, weight-level methods cannot detect *computational faults* that corrupt intermediate values during execution. In contrast, CSMR's output-level approach can detect some computational faults (see Appendix E).

4.4 Empirical runtime overhead analysis

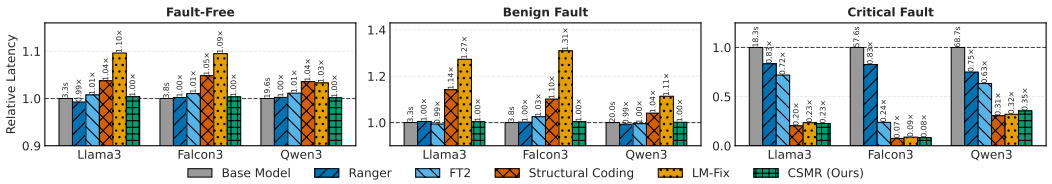

Figure 5: **CSMR incurs negligible latency overhead under fault-free and benign-fault conditions (0.1–0.4%).** For each model pair and fault setting, we report the *end-to-end latency* relative to applying no mechanism, averaged across 5 trials (95% CIs are within $\pm$0.002$\times$ across all bars).

We measure the average end-to-end latency of CSMR compared to each baseline; we also provide a theoretical analysis in Appendix C. For each model pair, we sample 100 inputs across benchmarks and record latency across five trials. Figure 5 compares the mean latency *relative* to the base model.

CSMR incurs negligible overhead under fault-free conditions. Because most inference runs encounter no faults, minimizing runtime overhead during fault-free execution is critical. Here, CSMR increases latency by at most 0.03 seconds (0.1–0.4% overhead), since detection requires one single NLL and JS divergence computation per generated token, and false positives are rare enough (see Appendix D) to have negligible impact. CSMR achieves latency comparable to or lower than Ranger and FT2, which have up to 1.2% overhead, while incurring much lower overhead than Structural Coding and LM-Fix (3.3–9.6%), whose detection mechanisms impose costs even without faults.

Under benign faults, CSMR's advantage grows. Most random faults are benign (see Appendix B), so an ideal resilience mechanism would ignore them entirely. CSMR achieves this by design: its discrepancy metric remains below τ for benign faults, so no recovery is triggered and overhead remains negligible (0.1–0.4%). Structural Coding and LM-Fix, by contrast, cannot distinguish benign from critical faults and incur the full repair cost for every fault, resulting in overheads of 4.1–31.0%. Ranger and FT2 are on par with CSMR, as their bounding operations have low latency.

Under critical faults, CSMR reduces latency by preventing costly generations. Counterintuitively, CSMR *reduces* end-to-end latency by 2.9–12.5$\times$ relative to the unprotected base model, because many critical faults produce repetitive outputs that hit the token limit. CSMR has similar latency to Structural Coding and LM-Fix—as all three successfully repair most corrupted weights—while substantially outperforming Ranger and FT2, which fail to prevent the costly generations.

4.5 Memory overhead analysis

As memory overhead is highly deployment-dependent, we provide a general analysis. First, the overhead of CSMR's detection logic is negligible: it stores the parameters (τ, w, and the normalization statistics), at most w discrepancy scores, and one 64-bit hash per weight tensor in M. Even for our largest model pair, the total is on the order of kilobytes. Second, the dominant memory cost is the draft model itself. Memory usage (in bytes) can be approximated as $P \times 2 \times (1 + A)$, where P is the parameter count and A denotes fractional additional memory for activations and the KV cache. Under typical settings, A can reach up to $\sim$30% (BentoML; Spheron), yielding a total footprint of 2.6–10.4 GB for our 1–4B parameter draft models, corresponding to a 10.0–12.5% increase over inference on the target model. While this cost is nontrivial, it is already assumed in deployments that use speculative decoding, and we discuss strategies to reduce it in Appendix F.

For the baselines, Ranger, FT2, and LM-Fix incur smaller overhead (<2%), as they store only a few scalars needed for detection. Structural Coding's default configuration (Asgari Khoshouyeh et al., 2023) imposes an 11.1% static memory penalty, comparable to CSMR (10.0–12.5%). While the overhead of Structural Coding is purely for protection, CSMR enables speculative decoding.

4.6 Generalization across memory fault models

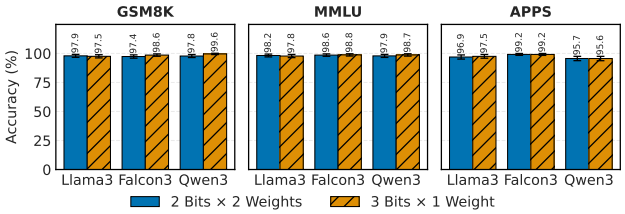

Figure 6: **CSMR generalizes to additional fault models**, achieving 95.6–99.6% accuracy under critical faults.

To evaluate whether CSMR generalizes beyond our primary single-weight 2-bit fault model, we test two additional fault models: *two-weight 2-bit*, which flips 2 bits in each of two independently selected weights, distributing corruption across the model, and *single-weight 3-bit*, which flips 3 bits in one weight—the minimum corruption that can potentially evade standard SEC-DED ECC detection. For both settings, we follow the sampling procedure in §4.2.

Figure 6 reports CSMR's accuracy under critical faults across all models and benchmarks: **CSMR remains resilient across diverse memory fault models**, recovering accuracy to 95.6–99.6%. Performance is also consistent, differing by at most 1.8 percentage points across model–benchmark pairs, indicating that it handles both distributed and multi-bit (>2) corruptions effectively.

4.7 Robustness to draft model faults

Here we evaluate CSMR when faults also affect the draft model. We consider two settings: *draft-model only*, where we inject a critical fault only into the draft model, and *both models*, where we independently sample critical faults for the draft and target models and inject them simultaneously. Figure 7 reports CSMR's TPR under critical single-weight 2-bit faults in both settings. We use TPR rather than accuracy because speculative decoding prevents the draft model from directly altering the target model's output; with a fault-free target model, output accuracy is therefore 100%.

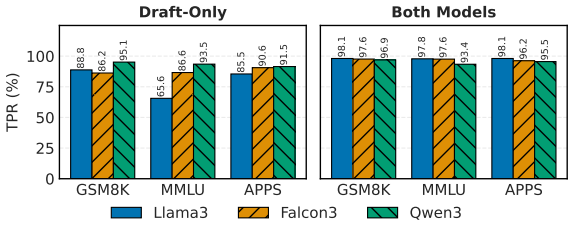

Figure 7: **CSMR remains effective under draft-model faults**, detecting most draft-only faults and maintaining high TPR when both models are corrupted.

CSMR remains largely effective under draft-model faults. Although draft-model-only faults do not affect the final output, we still detect them 65.6–95.1% of the time. This indicates that, despite CSMR being designed primarily to detect target-model corruption, its cross-scale discrepancy signal is sensitive to divergence caused by either model. TPR is even stronger when both models are corrupted (93.4–98.1%), comparable to the results in §4.3. Thus, draft-model faults do not substantially impact CSMR's effectiveness, even when combined with a critical target-model fault.

5 Conclusion

This work shows that numerical correctness is not always aligned with output correctness in LLM inference under hardware faults: substantial numerical perturbations can leave outputs intact, while small deviations may cause output divergence. Motivated by this mismatch, we propose Cross-Scale Model Resilience (CSMR), a decoding-level protection mechanism that detects faults through output divergence between a target model and a lightweight draft model, and triggers recovery only when faults affect model outputs. Across model families spanning 8–32B parameters and three benchmarks, CSMR restores accuracy under critical 2- and 3-bit memory faults from 0% to 95.6–99.6% with only 0.1–0.4% latency overhead. By shifting protection to the output level, CSMR offers scalable, task-agnostic resilience without full-model replication or infrastructure changes.

AI USE STATEMENT

In this work, we used generative AI tools to discuss refinements to aspects of the method and experimental design, implement methods, write code for experiments and figures, identify relevant literature, and draft and edit portions of the paper for clarity and readability. The core research ideas and methodological decisions originated with the authors. We reviewed and validated AI-assisted outputs before use, verified the existence and relevance of AI-found literature, and proofread AI-proposed paper edits before incorporating them. We did not use generative AI for any other tasks on ICLR's required-disclosure list. We take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

We release the artifacts used to obtain our main experimental results. Furthermore, we describe the hardware, software, hyperparameters, datasets, and models used in our work in the main paper and Appendix A. We believe these steps are sufficient to reproduce our work. Our code is available at: `https://anonymous.4open.science/r/CSMR-ICLR-2027`.

REFERENCES

Udit Kumar Agarwal, Abraham Chan, and Karthik Pattabiraman. Resilience assessment of large language models under transient hardware faults. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 659–670. IEEE, 2023. URL `https://doi.org/10.1109/ISSRE59848.2023.00052`.

Austin Appleby. smhasher. `https://github.com/aappleby/smhasher`.

Ali Asgari Khoshouyeh, Florian Geissler, Syed Qutub, Michael Paulitsch, Prashant Nair, and Karthik Pattabiraman. Structural coding: A low-cost scheme to protect cnns from large-granularity memory faults. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701092. doi: 10.1145/3581784.3607084. URL `https://doi.org/10.1145/3581784.3607084`.

BentoML. Calculating GPU memory for serving LLMs. `https://bentoml.com/llm/getting-started/calculating-gpu-memory-for-llms`.

Nandhini Chandramoorthy, Karthik Swaminathan, Martin Cochet, Arun Paidimarri, Schuyler Eldridge, Rajiv V Joshi, Matthew M Ziegler, Alper Buyuktosunoglu, and Pradip Bose. Resilient low voltage accelerators for high energy efficiency. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 147–158. IEEE, 2019. URL `https://doi.org/10.1109/hpca.2019.00034`.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023. URL `https://doi.org/10.48550/ARXIV.2302.01318`.

Zitao Chen, Guanpeng Li, and Karthik Pattabiraman. A low-cost fault corrector for deep neural networks through range restriction. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 1–13. IEEE, 2021. URL `https://doi.org/10.1109/dsn48987.2021.00018`.

Zachary Coalson, Jeonghyun Woo, Chris S. Lin, Joyce Qu, Yu Sun, Shiyang Chen, Lishan Yang, Gururaj Saileshwar, Prashant Nair, Bo Fang, and Sanghyun Hong. Prisonbreak: Jailbreaking large language models with at most twenty-five targeted bit-flips, 2025. URL `https://arxiv.org/abs/2412.07192`.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL `https://doi.org/10.48550/ARXIV.2110.14168`.

Sanjay Das, Swastik Bhattacharya, Souvik Kundu, Shamik Kundu, Anand Menon, Arnab Raha, and Kanad Basu. Attentionbreaker: Adaptive evolutionary optimization for unmasking vulnerabilities in LLMs through bit-flip attacks. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=2ekgTdBOZo.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3029–3051, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.183. URL https://doi.org/10.18653/v1/2023.emnlp-main.183.

Harish Dattatraya Dixit, Sneha Pendharkar, Matt Beadon, Chris Mason, Tejasvi Chakravarthy, Bharath Muthiah, and Sriram Sankar. Silent data corruptions at scale, 2021. URL https://doi.org/10.48550/ARXIV.2102.11245.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pretrained transformers. *arXiv preprint arXiv:2210.17323*, 2022. URL https://doi.org/10.48550/ARXIV.2210.17323.

Vinicius Fratin, Daniel Oliveira, Caio Lunardi, Fernando Santos, Gennaro Rodrigues, and Paolo Rech. Code-dependent and architecture-dependent reliability behaviors. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 13–26. IEEE, 2018. URL https://doi.org/10.1109/dsn.2018.00015.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models, 2024. URL https://doi.org/10.48550/ARXIV.2407.21783.

Yanan Guo, Liang Liu, Yueqiang Cheng, Youtao Zhang, and Jun Yang. Modelshield: A generic and portable framework extension for defending bit-flip based adversarial weight attacks. In *2021 IEEE 39th International Conference on Computer Design (ICCD)*, pp. 559–562. IEEE, 2021. URL https://doi.org/10.1109/iccd53106.2021.00090.

Richard W Hamming. Error detecting and error correcting codes. *The Bell system technical journal*, 29(2):147–160, 1950. URL https://doi.org/10.1002/j.1538-7305.1950.tb00463.x.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with APPS. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021a. URL https://openreview.net/forum?id=sD93GOzH3i5.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021b. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygGQyrFvH.

Mu-Yue Hsiao. A class of optimal minimum odd-weight-column sec-ded codes. *IBM Journal of Research and Development*, 14(4):395–401, 1970. URL https://doi.org/10.1147/rd.144.0395.

HuggingFace. Optimizing llms for speed and memory. URL https://huggingface.co/docs/transformers/en/llm_tutorial_optimization.

Mojan Javaheripi and Farinaz Koushanfar. Hashtag: Hash signatures for online detection of fault-injection attacks on deep neural networks, 2021. URL https://doi.org/10.1109/iccad51958.2021.9643556.

Mojan Javaheripi, Jung-Woo Chang, and Farinaz Koushanfar. Acchashtag: Accelerated hashing for detecting fault-injection attacks on embedded neural networks. *J. Emerg. Technol. Comput. Syst.*, 19(1), 2022. ISSN 1550-4832. doi: 10.1145/3555808. URL https://doi.org/10.1145/3555808.

Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. *ACM SIGARCH Computer Architecture News*, 42(3):361–372, 2014. URL https://doi.org/10.1109/isca.2014.6853210.

S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951. doi: 10.1214/aoms/1177729694. URL https://doi.org/10.1214/aoms/1177729694.

Jinqi Lai, Wensheng Gan, Jiayang Wu, Zhenlian Qi, and Philip S. Yu. Large language models in law: A survey. *AI Open*, 5:181–196, 2024. ISSN 2666-6510. URL https://doi.org/10.1016/j.aiopen.2024.09.002.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023. URL https://dl.acm.org/doi/10.5555/3618408.3619203.

Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W Keckler. Understanding error propagation in deep learning neural network (dnn) accelerators and applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–12, 2017. URL https://doi.org/10.1145/3126908.3126964.

Yu Li, Min Li, Bo Luo, Ye Tian, and Qiang Xu. Deepdyve: Dynamic verification for deep neural networks. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, pp. 101–112. ACM, October 2020. doi: 10.1145/3372297.3423338. URL http://dx.doi.org/10.1145/3372297.3423338.

Yuhang Liang, 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*, PPoPP '25, pp. 252–266, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400714436. doi: 10.1145/3710848.3710870. URL https://doi.org/10.1145/3710848.3710870.

J. Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991. URL https://doi.org/10.1109/18.61115.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration. In *MLSys*, 2024. URL https://doi.org/10.48550/ARXIV.2306.00978.

Haoxuan Liu, Vasu Singh, Michał Filipiuk, and Siva Kumar Sastry Hari. Alberta: Algorithm-based error resilience in transformer architectures. *IEEE Open Journal of the Computer Society*, 2024. URL https://doi.org/10.1109/ojcs.2024.3400696.

Dongning Ma, Fred Lin, Alban Desmaison, Joel Coburn, Daniel Moore, Sriram Sankar, and Xun Jiao. Dr. dna: Combating silent data corruptions in deep learning using distribution of neuron activations. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 239–252, 2024. URL https://doi.org/10.1145/3620666.3651349.

Andrew Milluzzi and Alan George. Exploration of TMR fault masking with persistent threads on Tegra GPU SoCs. In *2017 IEEE Aerospace Conference*, pp. 1–7. IEEE, 2017. URL https://doi.org/10.1109/aero.2017.7943882.

Zabir Al Nazi and Wei Peng. Large language models in healthcare and medical domain: A review. *Informatics*, 11(3), 2024. ISSN 2227-9709. URL https://doi.org/10.3390/informatics11030057.

Bin Nie, Devesh Tiwari, Saurabh Gupta, Evgenia Smirni, and James H Rogers. A large-scale study of soft-errors on gpus in the field. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 519–530. IEEE, 2016. URL https://doi.org/10.1109/hpca.2016.7446091.

NVIDIA. Nvidia tesla v100 gpu architecture, 2017. URL https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.

NVIDIA. Nvidia a100 tensor core gpu architecture, 2020. URL https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf.

Vladyslav Oles, Anna Schmedding, George Ostrouchov, Woong Shin, Evgenia Smirni, and Christian Engelmann. Understanding gpu memory corruption at extreme scale: The summit case study. In *Proceedings of the 38th ACM International Conference on Supercomputing*, pp. 188–200, 2024. URL https://doi.org/10.1145/3650200.3656615.

Tahmid Hasan Prato, Seijoon Kim, Lizhong Chen, and Sanghyun Hong. Hessian-aware training for enhancing dnns resilience to parameter corruptions. *arXiv preprint arXiv:2504.01933*, 2025. URL https://doi.org/10.48550/ARXIV.2504.01933.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018. URL https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Bit-flip attack: Crushing neural network with progressive bit search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1211–1220, 2019. URL https://doi.org/10.1109/iccv.2019.00130.

Lucas Roquet, Fernando Fernandes dos Santos, Paolo Rech, Marcello Traiola, Olivier Sentieys, and Angeliki Kritikakou. Cross-layer reliability evaluation and efficient hardening of large vision transformers models. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pp. 1–6, 2024. URL https://doi.org/10.1145/3649329.3655688.

Anna Schmedding, Lishan Yang, Adwait Jog, and Evgenia Smirni. Aspis: Lightweight neural network protection against soft errors. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 248–259. IEEE, 2024. URL https://doi.org/10.1109/issre62328.2024.00036.

scikit-learn developers. RobustScaler. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html.

SGLang Team. Logits processing. https://github.com/sgl-project/sglang/blob/cfab0ff/python/sglang/srt/layers/multimodal.py.

Qutub Syed Sha, Michael Paulitsch, Karthik Pattabiraman, Korbinian Hagn, Fabian Oboril, Cornelius Buerkle, Kay-Ulrich Scholl, Gereon Hinz, and Alois Knoll. Global clipper: Enhancing safety and reliability of transformer-based object detection models, 2024. URL https://doi.org/10.48550/ARXIV.2406.03229.

Charles W Slayman. Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations. *IEEE Transactions on Device and Materials Reliability*, 5 (3):397–404, 2005. URL https://doi.org/10.1109/tdmr.2005.856487.

Spheron. GPU Memory Requirements for LLMs: VRAM Calculator. https://www.spheron.network/blog/gpu-memory-requirements-llm/.

Guy L. Steele, Doug Lea, and Christine H. Flood. Fast splittable pseudorandom number generators. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '14, pp. 453–472, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450325851. doi: 10.1145/2660193.2660195. URL https://doi.org/10.1145/2660193.2660195.

Yu Sun, Zachary Coalson, Shiyang Chen, Hang Liu, Zhao Zhang, Sanghyun Hong, Bo Fang, and Lishan Yang. Demystifying the resilience of large language model inference: An end-to-end perspective. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '25, pp. 1127–1144, New York, NY, USA, 2025a. Association for Computing Machinery. ISBN 9798400714665. doi: 10.1145/3712285.3759803. URL https://doi.org/10.1145/3712285.3759803.

Yu Sun, Zhu Zhu, Cherish Mulpuru, Roberto Gioiosa, Zhao Zhang, Bo Fang, and Lishan Yang. Ft2: First-token-inspired online fault tolerance on critical layers for generative large language models. In *Proceedings of the 34th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '25, New York, NY, USA, 2025b. Association for Computing Machinery. ISBN 9798400718694. doi: 10.1145/3731545.3731570. URL https://doi.org/10.1145/3731545.3731570.

Ahmad Tahmasivand, Noureldin Zahran, Saba Al-Sayouri, Mohammed Fouda, and Khaled N Khasawneh. Lm-fix: Lightweight bit-flip detection and rapid recovery framework for language models. In *2025 IEEE 43rd International Conference on Computer Design (ICCD)*, pp. 432–440. IEEE, 2025. URL https://doi.org/10.1109/iccd65941.2025.00068.

Falcon-LLM Team. The falcon 3 family of open models, December 2024. URL https://huggingface.co/blog/falcon3.

Devesh Tiwari, Saurabh Gupta, George Gallarno, Jim Rogers, and Don Maxwell. Reliability lessons learned from gpu experience with the titan supercomputer at oak ridge leadership computing facility. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pp. 1–12, 2015. URL https://doi.org/10.1145/2807591.2807666.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. URL https://dl.acm.org/doi/10.5555/3295222.3295349.

Jialai Wang, Ziyuan Zhang, Meiqi Wang, Han Qiu, Tianwei Zhang, Qi Li, Zongpeng Li, Tao Wei, and Chao Zhang. Aegis: mitigating targeted bit-flip attacks against deep neural networks, 2023. URL https://dl.acm.org/doi/10.5555/3620237.3620368.

Shen Wang, Tianlong Xu, Hang Li, Chaoli Zhang, Joleen Liang, Jiliang Tang, Philip S. Yu, and Qingsong Wen. Large language models for education: A survey and outlook. *IEEE Signal Processing Magazine*, 42(6):51–63, 2025. URL https://doi.org/10.1109/MSP.2025.3594309.

Shibo Wang and Pankaj Kanwar. Bfloat16: The secret to high performance on cloud tpus, 2019. URL https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus.

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. Neural text generation with unlikelihood training. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJeYe0NtvH.

Hanxiang Xu, Shenao Wang, Ningke Li, Kailong Wang, Yanjie Zhao, Kai Chen, Ting Yu, Yang Liu, and Haoyu Wang. Large language models for cyber security: A systematic literature review. *ACM Trans. Softw. Eng. Methodol.*, September 2025a. ISSN 1049-331X. doi: 10.1145/3769676. URL https://doi.org/10.1145/3769676.

Haotian Xu, Qingsong Peng, Jie Shi, Huadi Zheng, Yu Li, and Cheng Zhuo. Silentstriker: Toward stealthy bit-flip attacks on large language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b. URL https://openreview.net/forum?id=e40dYCosQd.

Xinghua Xue, Cheng Liu, Feng Min, Tao Luo, and Yinhe Han. Approxabft: Approximate algorithm-based fault tolerance for neural network processing. *arXiv preprint arXiv:2302.10469*, 2023. URL https://doi.org/10.48550/ARXIV.2302.10469.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report, 2025. URL https://doi.org/10.48550/ARXIV.2505.09388.

Lishan Yang, Bin Nie, Adwait Jog, and Evgenia Smirni. Enabling software resilience in GPGPU applications via partial thread protection. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 1248–1259. IEEE, 2021. URL https://doi.org/10.1109/icse43902.2021.00114.

Qiang Zhang, Keyan Ding, Tianwen Lv, Xinda Wang, Qingyu Yin, Yiwen Zhang, Jing Yu, Yuhao Wang, Xiaotong Li, Zhuoyi Xiang, et al. Scientific large language models: A survey on biological & chemical domains. *ACM Computing Surveys*, 57(6):1–38, 2025. URL https://doi.org/10.1145/3715318.

Wenxin Zheng, Bin Xu, Jinyu Gu, and Haibo Chen. SAVE: Software-Implemented fault tolerance for model inference against GPU memory bit flips. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, pp. 1585–1604, 2025. URL https://dl.acm.org/doi/10.5555/3768039.3768133.

Zhu Zhu, Yu Sun, Dhatri Parakal, Bo Fang, Steven Farrell, Gregory H. Bauer, Brett Bode, Ian T. Foster, Michael E. Papka, William Gropp, Zhao Zhang, and Lishan Yang. Understanding the landscape of ampere gpu memory errors, 2025. URL https://doi.org/10.48550/ARXIV.2508.03513.

A Detailed experimental setup

We run all experiments using two machines. The first is equipped with a 48-core Intel Xeon processor, 768 GB of memory, and 8 NVIDIA A40 (48 GB) GPUs. The second has a 32-core AMD EPYC 9354P processor, 384 GB of memory, and 4 NVIDIA RTX PRO 6000 GPUs. We use only the first machine for computing latency to ensure consistent measurements and a fair comparison across methods; we use the second machine for sampling faults and non-latency evaluations. We implement CSMR with Python 3.10, PyTorch 2.7.0, and Transformers 4.51.1.

B Fault statistics

Model	GSM8K		MMLU		APPS		Total	
	B	C	B	C	B	C	B+C	% B
Llama3	4,520	519	3,908	515	3,043	407	12,912	88.8
Falcon3	4,013	558	4,823	561	4,234	550	14,739	88.7
Qwen3	4,495	572	4,168	539	3,622	523	13,919	88.3
Total	13,028	1,649	12,899	1,615	10,899	1,480	41,570	88.6

Table 2: **Benign (B) and critical (C) fault counts from our sampling procedure.** Benign faults are 6.9–8.7× more prevalent than critical faults across all model/benchmark pairs.

To characterize the prevalence of fault outcomes, we aggregate statistics from our offline fault sampling in Table 2. Of 41,570 randomly sampled single-weight 2-bit memory faults, 88.6% are benign while 11.4% are critical. These proportions are similar across models (within ±0.5%), suggesting stability across settings. Notably, **benign faults are 6.9–8.7× more prevalent than critical ones**.

C Runtime complexity analysis

As empirical latency depends on hardware and implementation details, we complement our analysis in §4.4 with a theoretical analysis of the runtime overhead under fault-free execution, assuming no false positives (which are exceptionally rare for CSMR; see Appendix D). Table 3 summarizes the theoretical overhead of CSMR and baseline resilience mechanisms with respect to fixed target model components, separated into pre-generation and per-token costs.

Ranger and FT2 apply activation bounding at every Transformer block, incurring $\mathcal{O}(L \cdot D_h)$ per-token overhead. Structural Coding and LM-Fix operate on all weight matrices before generation begins, making their pre-generation cost $\mathcal{O}(P)$. In contrast, **CSMR adds no pre-generation overhead, and its per-token cost does not scale with model size.** Its per-token cost is $\mathcal{O}(|\mathcal{V}|)$, determined by the vocabulary size rather than parameter count. This distinction is significant as vocabulary size typically does not scale with model size (Yang et al., 2025; Grattafiori et al., 2024; Team, 2024). Moreover, since latency itself scales with parameter count, CSMR's fixed $\mathcal{O}(|\mathcal{V}|)$ overhead constitutes a diminishing fraction of total per-token cost as models continue to scale.

Method	Before Generation	Per-Token
Ranger	–	$\mathcal{O}(L \cdot D_h)$
FT2	–	$\mathcal{O}(L \cdot D_h)$
Structural Coding	$\mathcal{O}(P)$	–
LM-Fix	$\mathcal{O}(P)$	–
CSMR (Ours)	–	$\mathcal{O}(\lvert\mathcal{V}\rvert)$

L: # of blocks; D_h: hidden dimension; P: # of parameters; $\mathcal{V}$: vocabulary.

Table 3: **CSMR's runtime overhead does not scale with model size.** The theoretical overhead under fault-free execution.

D Failure mode analyses

To understand the limitations of CSMR, we qualitatively analyze false positives and false negatives.

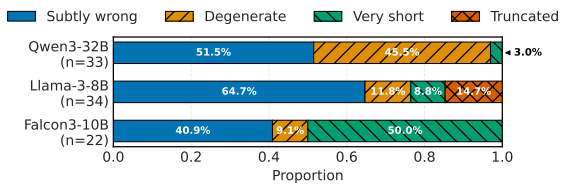

Figure 8: **Qualitative categorization of CSMR's false negatives by model.**

False negatives. CSMR produces 89 false negatives across 4,164 critical-fault cases (2.1%). We manually inspect each output and group them into four qualitative categories: *subtly wrong* outputs that preserve fluency while altering correctness, *degenerate* outputs (e.g., repetition loops), *very short* answers (a direct answer without any reasoning), and *truncated* outputs (no visible corruption but terminating before an answer). Figure 8 reports the overall distribution by model.

The dominant category is subtly wrong (53.9%), followed by degenerate (23.6%), very short (16.9%), and truncated (5.6%). Subtly wrong outputs may evade detection when the corrupted continuation remains a high-probability completion under both models, keeping the discrepancy signal below threshold despite the semantic error. Degenerate outputs could reflect a related effect: LLMs are prone to repetition and degenerate generation (Holtzman et al., 2020; Welleck et al., 2020), so when the target enters such a state, the draft may follow, producing low-divergence outputs. Very short answers offer few tokens for windowed aggregation to accumulate signal, limiting CSMR's sensitivity in this regime. Truncated outputs are likely not a defect of CSMR but of our evaluation setting, as these generations may have terminated correctly under a larger token budget.

False positives. Across all 8,328 benign and fault-free executions, CSMR produces only 8 false positives (0.1%), too few for meaningful qualitative analysis. However, this low rate clearly indicates that benign model disagreement stays well-separated from fault-induced divergence.

E Effectiveness of CSMR against computational faults

Following Sun et al. (2025a), we consider a 2-bit computational fault model in which each generation is subjected to 2 random bit-flips applied to a randomly selected activation and forward pass. Using the same method as §4.2, we identify one critical and one benign fault per example. Because computational faults exhibit distinct propagation patterns (Sun et al., 2025a), the optimal configuration identified in Appendix G is unlikely to transfer. Therefore, we evaluate CSMR across varying configurations by reporting TPR and FPR for different values of τ and w. Figure 9 shows the results for Llama3 on GSM8K; we omit FPR under fault-free execution, as it is near 0 for all values.

CSMR can also mitigate computational faults. Our optimal memory-fault configuration ($\tau = 99.5$, $w = 10$) achieves only 25% TPR, as it is tuned to capture faults whose effects persist across multiple token positions rather than a single forward pass. However, by lowering the stability threshold and window size, CSMR achieves up to 76% TPR with $\leq$2% FPR. Since prior work has shown that computational faults rarely induce incorrect answers (Sun et al., 2025a), this detection rate implies that CSMR can preserve correctness in the vast majority of all cases.

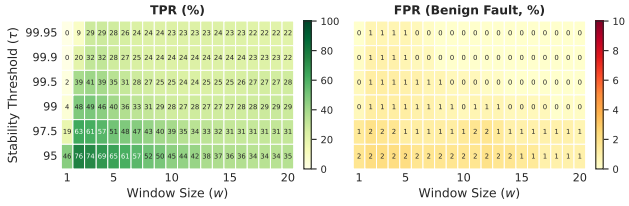

Figure 9: **CSMR achieves up to 76% TPR with $\leq$2% FPR against computational faults** for GSM8K on Llama3.

F Minimizing memory overhead

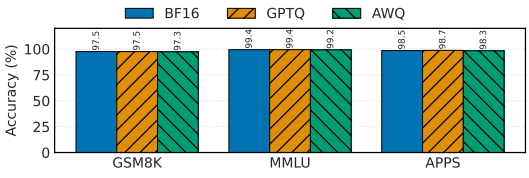

Figure 10: **Quantization has a negligible impact on CSMR's effectiveness.** The accuracy across benchmarks and quantization schemes for Falcon3.

To reduce CSMR's memory overhead, we explore *quantization*, which represents model weights at reduced precision. We evaluate draft models deployed in 4-bit integer precision, which reduces each parameter to 0.5 bytes and lowers the memory overhead from §4.5 to 4.2–5.3%. We consider two widely used schemes, GPTQ (Frantar et al., 2022) and AWQ (Lin et al., 2024). Using official quantized checkpoints for Falcon3-1B-Instruct, we report accuracy under critical faults for CSMR using the Falcon3 model pair in Figure 10.

CSMR is compatible with quantized draft models: across benchmarks, both schemes change accuracy by at most 0.2 percentage points, indicating a negligible impact on detection performance. This suggests that the draft model can operate at substantially lower precision than the target model while still providing strong output-aware discrepancy signals.

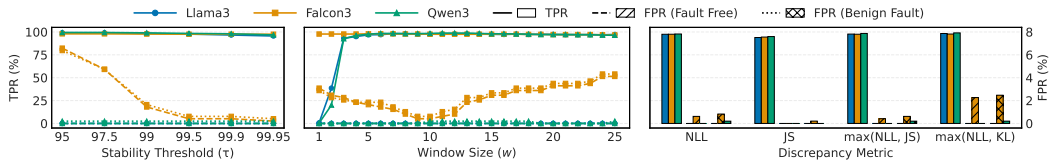

Figure 11: **CSMR's default configuration (τ = 99.5, w = 10, max(NLL, JS) metric) achieves a strong TPR/FPR trade-off across all models without per-model tuning.** We analyze CSMR's sensitivity to different configurations by sweeping over the stability threshold τ, window size w, and discrepancy metric construction, reporting TPR (fraction of correctly detected critical faults) and FPR (fraction of incorrectly flagged fault-free or benign-fault executions).

G Sensitivity Analysis

We evaluate the sensitivity of CSMR to its key configuration choices. Because accuracy alone cannot measure false positives, we instead recompute the composite metric D offline from the raw per-token completions generated during our main experiments and report TPR and FPR (defined in §4.1). Figure 11 presents results on GSM8K across all models.

Stability threshold (τ). We vary τ over the $\{95, 97.5, 99, 99.5, 99.9, 99.95\}$-th percentiles of the offline calibration distribution. Increasing τ reveals a model-dependent trade-off: Falcon3's FPR drops substantially (6.4–6.6% to 0.2–0.4%), reflecting a heavier-tailed benign distribution, while Llama3 and Qwen3 stay near zero (0.0–0.2%) throughout, indicating well-separated fault and benign discrepancies. TPR decreases only modestly across all models (97.9–99.4% to 95.6–97.5%). We adopt the 99.5-th percentile, which consistently achieves $\geq$97.5% TPR at $\leq$0.6% FPR.

Window size (w). We vary w over $\{1, 2, \ldots, 25\}$ and find that its effect is highly model-dependent. For Llama3 and Qwen3, TPR increases sharply from 0.4–0.6% at $w = 1$ to 96.4–97.3% at $w = 25$, while FPR remains $\leq$0.2%. This indicates that fault-induced discrepancy in these models accumulates primarily across consecutive tokens, making windowed aggregation essential. For Falcon3, TPR stays consistently high (97–98%) across all window sizes, but FPR follows a U-shape: 2.9–3.1% at $w = 1$, 0.4–0.6% at $w = 10$, and 4.1–4.3% at $w = 25$. This suggests that narrow windows incorporate too little context to smooth out benign disagreement, while wide ones dilute the fault signal. We adopt $w = 10$, which achieves near-optimal TPR with low FPR across all models.

Discrepancy metrics. We evaluate three alternatives to D: NLL only, JS divergence only, and replacing JS with KL divergence. The individual components show complementary strengths: NLL achieves higher TPR (97.5–97.8%) than JS divergence (94.0–95.0%), but at the cost of elevated FPR (0.0–0.8% versus 0.0–0.2%). Our composite metric ($\max$(NLL, JS)) outperforms both—achieving 97.5–98.4% TPR at only 0.0–0.6% FPR—confirming that neither component alone is sufficient. Substituting KL for JS yields comparable TPR (97.7–99.0%) but increases FPR by up to 2.5%, likely because KL divergence is unbounded and asymmetric, amplifying benign distributional differences.