# Fault Robustness of Custom Floating-Point and Integer Formats: Datatype Selection as a Reliability-Aware Compression Decision

**R S Haripriya** [1] **Jaynarayan T Tudu** [2]

## Abstract

Deploying neural networks on resource-constrained edge devices requires both memory-efficient quantization and robustness to hardware-induced faults. We present a unified fault robustness study across fourteen low-precision datatypes spanning custom Float16, Float8, and integer representations under random multi-bit DRAM disturbance faults. Robustness is evaluated through top-1 accuracy degradation on MobileNetV2, EfficientNet-B0, and ResNet-18 across CIFAR-10, CIFAR-100, and Tiny ImageNet. Our results show that floating-point fault sensitivity is dominated by exponent-field width, with wider exponents causing severe error amplification under bit flips. Among all evaluated formats, E4M11 consistently provides the best trade-off between clean accuracy and fault robustness: it incurs only a $0.23\%$ average clean-accuracy drop relative to FP32 — on par with FP16 and BF16 — yet limits worst-case exponent-fault accuracy loss to $6.1\%$ on average, versus $34.6\%$ for FP16 and $36.2\%$ for BF16, a $5.7\times$/$5.9\times$ robustness gain. Although INT16 is structurally immune to exponent faults, it delivers $\sim$10.7 dB lower Signal-to-Quantisation-Noise Ratio (SQNR) than E4M11 with no compression benefit; INT8 avoids exponent amplification but suffers $\sim$58.8 dB lower SQNR and larger accuracy degradation on harder tasks. E4M11 thus uniquely combines near-FP32 accuracy, the highest SQNR ($\sim$79.8 dB) among all formats studied, and superior fault tolerance, establishing datatype selection as a *reliability-aware* design mechanism that requires no retraining or error-correction hardware.

[1]Department of Electrical Engineering, IIT Tirupati, Tirupati, India [2]Department of Computer Science and Engineering, IIT Tirupati, Tirupati, India. Correspondence to: R S Haripriya <ee22d004@iittp.ac.in>, Jaynarayan T Tudu <jtt@iittp.ac.in>.

*AdaptFM: Resource-Adaptive Foundation Model Inference Workshop, co-located with ICML 2026, Seoul, South Korea.*

## 1. Introduction

Deep neural networks (DNNs) are increasingly deployed on resource-constrained edge devices for safety-critical applications such as medical diagnosis (Abràmoff et al., 2018), autonomous navigation (Bojarski et al., 2016), and disaster-response systems (Gupta et al., 2019), where limited memory and energy budgets demand efficient local inference (Canziani et al., 2016). Post-training quantization (PTQ) addresses this challenge by converting FP32 weights into lower-precision representations without retraining (Nagel et al., 2020; Gholami et al., 2022). However, low-cost edge hardware is also vulnerable to hardware-induced memory faults, including DRAM disturbance errors that silently flip stored weight bits without raising runtime exceptions (Baumann, 2005; Kim et al., 2014; Lin et al., 2024). Since quantized weights form the stored representation used during inference, such faults can directly corrupt model predictions.

The impact of hardware faults depends strongly on the underlying numeric representation. In floating-point formats, perturbation of exponent bits can cause severe error amplification, whereas integer formats are not subject to this effect due to the absence of exponent fields. Existing work has studied fault robustness primarily within isolated datatype families, focusing either on fixed-point accelerators (Li et al., 2017; Reagen et al., 2018) or adversarial bit-flip attacks (Rakin et al., 2019; He et al., 2020). Unified studies comparing custom floating-point and integer formats under a common fault-injection framework remain limited.

In this work, we present a systematic fault-robustness study across fourteen low-precision datatypes spanning custom Float16, Float8, and integer representations. We evaluate MobileNetV2, EfficientNet-B0, and ResNet-18 on CIFAR-10, CIFAR-100, and Tiny ImageNet under random single and multi-bit DRAM disturbance faults of up to eight simultaneous bit flips. Figure 1 illustrates the overall pipeline, where FP32 weights are quantized into low precision datatypes, subjected to memory faults, and evaluated based on the resulting inference degradation. Since E4M11 emerges as the most effective trade-off between clean accuracy and fault robustness, we additionally define explicit FP32-to-E4M11 quantization and dequantization

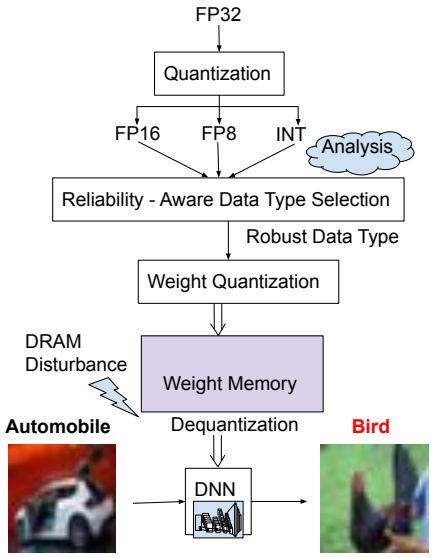


*Figure 1.* Overview of reliability-aware datatype selection for quantized neural networks under DRAM disturbance faults.

procedures, as such custom datatypes are not directly available in standard deep learning frameworks.

**Contributions.**

- We present a unified fault-robustness analysis spanning custom Float16, Float8, and integer datatypes under a consistent single and multi-bit fault injection framework.
- We identify exponent-field width as the dominant factor governing floating-point fault sensitivity, with wider exponents producing substantially larger error amplification under bit flips.
- We show that **E4M11** is the Pareto-optimal 16-bit format, it incurs only a $0.23\%$ average clean-accuracy drop relative to FP32, on par with FP16 and BF16, while reducing worst-case exponent fault accuracy loss by $5.7\times$ and $5.9\times$ over FP16 and BF16 respectively, while giving $\sim$10.7 dB and $\sim$58.8 dB higher SQNR than INT16 and INT8.

The remainder of the paper is organised as follows. Section 2 reviews related work; Section 3 describes datatype parameterization; Section 4 presents the fault model and injection procedure; and Section 5 discusses experimental results and deployment implications.

## 2. Background and Related Work

**Low-precision quantization formats.** IEEE Float16 (E5M10) and BFloat16 (E8M7) are the dominant 16-bit inference formats; BF16 was designed to preserve FP32 dynamic range via a wide exponent field (Kalamkar et al., 2019; Micikevicius et al., 2018), a property that, as we show, also increases fault sensitivity. 8-bit formats E4M3 and E5M2 offer further compression at reduced range (Micikevicius et al., 2022). On the integer datatypes, INT8 is the standard edge format due to hardware simplicity (Jacob et al., 2018), and INT4 enables aggressive compression (Banner et al., 2019). Additionally, integer formats contain no exponent field and therefore cannot exponentially amplify bit flips - a structural distinction that fundamentally shapes fault propagation and motivates our cross-family evaluation.

**Hardware faults and DNN resilience.** Particle-induced DRAM soft errors are an established reliability concern in deployed systems (Baumann, 2005). Kim et al. (2014) showed that DRAM disturbance faults are exploitable without privileged access, and Lin et al. (2024) extended this to GPU memories through RowHammer-style attacks, reporting up to eight simultaneous bit flips, which we adopt as the upper bound in our fault model. Li et al. (2017) identified high-magnitude weight perturbations as the dominant source of inference failure in fixed-point accelerators, while Reagen et al. (2018) showed that excessive numeric range can amplify soft-error damage, since a single exponent bit flip in a wide exponent field may induce disproportionately large weight perturbations. This observation directly motivates our systematic sweep of exponent-field width. Adversarial bit-flip attacks (Rakin et al., 2019; He et al., 2020) are related but distinct, as they target deliberately chosen flips rather than the random hardware-induced faults studied here, which more closely reflect real edge conditions.

**Fault-aware defences and their limitations.** Fault-aware retraining methods such as Ranger (Chen et al., 2021) reduce accuracy degradation by constraining activations but require training data and additional computation, which are often infeasible at the edge. ECC memory corrects single-bit errors but incurs additional hardware overhead (Hamming, 1950; Reagen et al., 2018). We explore a zero-cost approach in which datatype selection during quantization directly affects the fault resilience. The evaluations spanning custom Float16, Float8, and integer formats under a unified single and multi-bit fault injection framework remain limited.

## 3. Low-Precision Datatypes

We evaluate fourteen low-precision datatypes spanning custom Float16, Float8, and integer representations. The balance between exponent range and mantissa precision determines both representational fidelity and sensitivity to hardware-induced faults. Wider exponent fields increase the dynamic range but can amplify the effect of exponent-bit perturbations, whereas narrower exponents limit worst-case fault amplification at the cost of reduced numeric range. Among the evaluated formats, E4M11 emerges as a strong

trade-off between fault resilience, representational fidelity, and clean accuracy. This section formalizes the evaluated datatype families and defines the storage and reconstruction procedures used for E4M11. Tables 1 and 2 summarise the evaluated formats. All custom Float16 formats use one sign bit and satisfy $E + M = 15$, with exponent bias $\text{bias} = 2^{E-1}-1$, the subnormal numbers are flushed to zero, and the all-ones exponent pattern is reserved for $\pm\infty$/NaN.

*Table 1.* Custom 16-bit floating-point formats.

| Format | $E$ | $M$ | Bias | Max Finite Value |
|---|---|---|---|---|
| E2M13 | 2 | 13 | 1 | $\approx 4.0$ |
| E3M12 | 3 | 12 | 3 | $\approx 16.0$ |
| **E4M11** | **4** | **11** | **7** | $\approx 256$ |
| E5M10 (FP16) | 5 | 10 | 15 | $\approx 6.55 \times 10^4$ |
| E6M9 | 6 | 9 | 31 | $\approx 4.29 \times 10^9$ |
| E7M8 | 7 | 8 | 63 | $\approx 1.84 \times 10^{19}$ |
| E8M7 (BF16) | 8 | 7 | 127 | $\approx 3.39 \times 10^{38}$ |

*Table 2.* Custom 8-bit floating-point formats.

| Format | $E$ | $M$ | Bias | Max Finite Value |
|---|---|---|---|---|
| F8E3M4 | 3 | 4 | 3 | $\approx 15.5$ |
| F8E4M3 | 4 | 3 | 7 | $\approx 240$ |
| F8E5M2 | 5 | 2 | 15 | $\approx 5.73 \times 10^4$ |

### 3.1. Floating-Point Weight Storage

Floating-point formats store weights *directly* as bit patterns. Given a value $w$, the sign bit $s$, biased exponent $e_s$, and integer mantissa $m$ are packed into a fixed-width word:

$$b = (s \ll (E + M)) \mid (e_s \ll M) \mid m \tag{1}$$

Reconstruction requires only bit extraction and the standard floating-point decode as given in Equation 2.

$$\hat{w} = (-1)^s \left(1 + \frac{m}{2^M}\right) 2^{\,e_s - \text{bias}} \tag{2}$$

No scale factor, zero-point, or calibration data is required.

### 3.2. Integer Quantization and Dequantization

Integer formats require explicit quantization. Per-tensor symmetric quantization maps FP32 weights to signed two's-complement integers as presented in Equation 3.

$$w_q = \text{clip}\Big(\Big\lfloor \frac{w}{s} \Big\rceil,\ -2^{b-1},\ 2^{b-1}-1\Big), \qquad s = \frac{\max|w|}{2^{b-1}-1} \tag{3}$$

Where $b$ is the bit width and $\lfloor\cdot\rceil$ denotes nearest-integer rounding. Dequantization recovers the FP32 approximation as $\hat{w} = w_q \times s$. The scale factor $s$ must be stored alongside the integer codes and reapplied at every load.

**Example** ($w$=3.14, INT8, layer max = 4.0). $s$=4.0/127≈0.0315; $w_q$=100; $\hat{w}$=3.15 (relative error ≈0.31%).

### 3.3. E4M11: Proposed Format and Conversion

Among all fourteen formats evaluated, **E4M11** achieves a strong trade-off between fault resilience, representational fidelity, and clean accuracy, as demonstrated in Section 5. Since E4M11 is not natively available in standard deep learning frameworks such as PyTorch or NumPy, we define its encoding and reconstruction procedures explicitly below.

E4M11 uses 1 sign bit, 4 exponent bits (bias = 7), and 11 mantissa bits:


Its maximum finite value $(2 - 2^{-11}) \times 2^7 \approx 256$ covers the weight range of all evaluated models. The 11-bit mantissa yields the highest SQNR (∼79.8 dB versus ∼68.8 dB for INT16 and ∼20.7 dB for INT8). A bit flip at the exponent MSB multiplies the weight magnitude by $2^{\pm 2^{E-1}}$, and the worst-case perturbation factor is $\rho_{E-1} = 2^{2^{E-1}} - 1 = 255$ for E4M11, versus 65,535 for FP16 and $2^{128} - 1$ for BF16.

**Encoding** (FP32 → uint16). Given weight $w$:

$$\begin{aligned} e &= \text{clamp}(\lfloor \log_2 |w| \rfloor,\ -6,\ 7) \\ m &= \text{round}\Big(\Big(\tfrac{|w|}{2^e} - 1\Big) \cdot 2^{11}\Big) \\ &\quad \text{(if } m \geq 2^{11}\text{: set } e \leftarrow e{+}1,\ m \leftarrow 0) \\ e_s &= e + 7 \\ b &= (s \ll 15) \mid (e_s \ll 11) \mid m \end{aligned} \tag{4}$$

where $s = 1$ if $w < 0$, else $s = 0$.

**Decoding** (uint16 → FP32):

$$\hat{w} = (-1)^s \left(1 + \frac{m}{2^{11}}\right) 2^{\,e_s - 7} \tag{5}$$

**Example** ($w = 3.14$). $3.14 = 1.10010001111\ldots_2 \times 2^1$; $s$=0, $e_s$=8=$(1000)_2$, $m$=$(10010001111)_2$. Packed: `0 1000 10010001111` = `0x448F`; $\hat{w} \approx 3.1396$ (error 0.0127%, versus 0.31% for INT8).

## 4. Fault Model and Injection Procedure

### 4.1. Fault Model

We model DRAM disturbance errors as random bit flips in stored weight memory, following the RowHammer-style fault characterisation of Lin et al. (2024), in which up to eight simultaneous bit flips were observed per fault event. Faults are injected at inference time into the quantized

weight representation, before dequantization and forward pass. No fault detection or correction mechanism is assumed.

### 4.2. Fault Groups

To isolate the contribution of each bit field, we partition the bits of each format into four disjoint groups:

- **Group 1 – Exponent bits**: all $E$ exponent bits (positions $M$ to $M+E-1$). For integer formats, this group is empty by construction.
- **Group 2 – Sign bit**: the single sign bit (position $E+M$).
- **Group 3 – Mantissa / Value MSB**: the upper $\lceil M/2 \rceil$ mantissa bits. For integer formats, the upper half of the value field (excluding the sign bit).
- **Group 4 – Mantissa / Value LSB**: the lower $\lfloor M/2 \rfloor$ mantissa bits, or lower half of the integer value field.

### 4.3. Injection Procedure

For each fault group, bit-flip count $k \in \{1, \ldots, 8\}$, format, and model–dataset combination, we perform 100 independent trials. In each trial, $k$ weights are selected uniformly at random from the full weight tensor. For each selected weight, one bit is chosen uniformly at random from the target group and flipped via XOR. The perturbed weights are then dequantized and the model is evaluated on the full 10,000-sample test set. Mean and standard deviation of Top-1 accuracy are recorded across trials.

Formats whose post-quantization clean accuracy falls more than 30% below the FP32 baseline are excluded from fault injection because the model has already experienced substantial performance degradation. These correspond to E2M13, E3M12, F8E3M4, and INT2 on most configurations.

## 5. Results

### 5.1. Clean Accuracy After Quantization

Figure 2 shows clean PTQ accuracy for all fourteen formats across ResNet-18, EfficientNet-B0, and MobileNetV2 on CIFAR-10. A sharp phase transition appears in the Float16 family, where E2M13 and E3M12 reduce Top-1 accuracy to near-random performance (10%) due to insufficient dynamic range, whereas all formats with $E \geq 4$ remain within 0.1–0.2% of the ResNet-18, EfficientNet-B0, and MobileNetV2 FP32 baselines on CIFAR-10 (96.55%, 94.0%, 92.58% respectively).

Among Float8 formats, F8E3M4 reaches near-random accuracy for all three models on CIFAR-10. F8E4M3 remains close to FP32 accuracy (96.34%, 92.25%, 82.90% for ResNet-18, EfficientNet-B0, and MobileNetV2, corresponding to drops of 0.21%, 1.75%, and 9.68% from their FP32 baselines respectively), whereas F8E5M2 remains effective only for ResNet-18 (96.3%, drop of 0.25%) and degrades substantially for EfficientNet-B0 (70.93%, drop of 23.07%) and MobileNetV2 (40.16%, drop of 52.42%), indicating architecture-specific sensitivity to limited dynamic range.

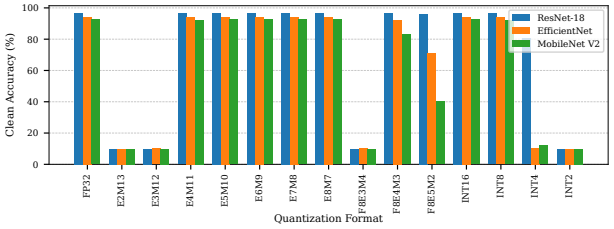


*Figure 2.* Clean PTQ accuracy on CIFAR-10 across all fourteen formats.

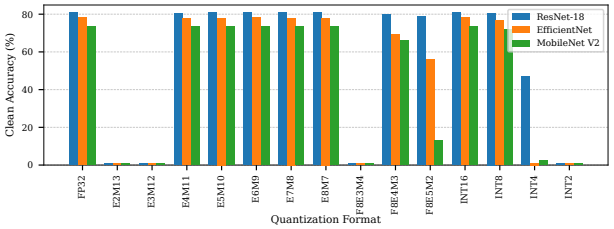


*Figure 3.* Clean PTQ accuracy on CIFAR-100 across all fourteen formats.

INT16 and INT8 remain close to FP32 accuracy on CIFAR-10 (96.55%, 94.0%, 92.58% and 96.52%, 93.79%, 92.05% respectively). INT4 shows a moderate reduction for ResNet-18 (80.47%) but reaches near-random performance for EfficientNet-B0 (10.51%) and MobileNetV2 (12.04%). INT2 reaches near-random accuracy across all three models.

The similar qualitative behaviour is observed on CIFAR-100 (Figure 3) and Tiny ImageNet (Figure 4). As dataset complexity increases, the set of viable formats narrows further, Float8 formats degrade substantially for EfficientNet-B0 and MobileNetV2, while INT4 reaches near-random performance for EfficientNet-B0 and MobileNetV2 on CIFAR-100 (1.07%, 2.43%) and for all three models on Tiny ImageNet, although ResNet-18 retains partial accuracy under INT4 on CIFAR-100 (47.15%). INT2 reaches near-random accuracy across both datasets. On Tiny ImageNet, INT16 consistently remains close to FP32, while INT8 degrades notably for EfficientNet-B0 (64.87% vs 68.86%) and MobileNetV2 (51.53% vs 60.75%). Only Float16 formats with $E \geq 4$ and INT16 consistently remain close to FP32 accuracy across all three datasets.

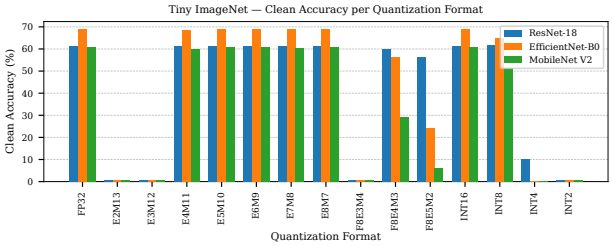


*Figure 4.* Clean PTQ accuracy on Tiny ImageNet across all fourteen formats.

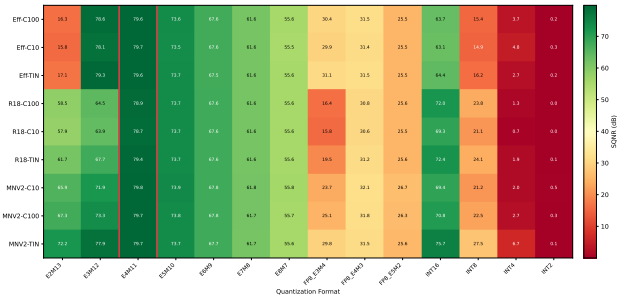


*Figure 5.* SQNR (dB) measured over convolutional and fully connected weights. E4M11 achieves the highest representational fidelity across all nine model-dataset combinations (78.7-79.8 dB). Formats left of E4M11 provide insufficient dynamic range, whereas formats right of E4M11 trade mantissa precision for largely unused exponent range.

### 5.2. Weight Representation Fidelity (SQNR)

Before fault injection, we evaluate representational fidelity using Signal-to-Quantisation-Noise Ratio (SQNR) measured over convolutional and fully connected weight tensors. For a weight tensor $w$ quantized to $\hat{w}$, SQNR is defined as:

$$\mathrm{SQNR} = 10 \log_{10} \left( \frac{\sum_i w_i^2}{\sum_i (w_i - \hat{w}_i)^2} \right) \text{ dB} \qquad (6)$$

where higher SQNR indicates lower quantisation error, and infinite SQNR corresponds to lossless storage.

Figure 5 summarises the results. **E4M11 consistently achieves the highest SQNR across all nine model-dataset combinations** (78.7-79.8 **dB)**, exceeding FP16/E5M10 (73.5-73.9 dB) by approximately $+5$ dB on average and INT16 (63.1-75.7 dB) by $+10$ dB. INT8 reaches only 14.9-27.5 dB, while INT4 and INT2 remain below 7 dB.

Although E2M13 and E3M12 achieve moderate SQNR in some cases due to their large mantissas, their limited exponent range causes high-magnitude weights to saturate, reducing inference accuracy to near-random performance. This indicates that adequate dynamic range is as important as mantissa precision for reliable weight storage, and empirically suggests that formats with fewer than four exponent bits are insufficient for the evaluated workloads.

### 5.3. Fault Sensitivity: Exponent Bits

Figures 6–8 show mean accuracy drop under exponent-bit faults (Group 1) as a function of bit-flip count. Sign-bit and mantissa results are summarised in Section 5.4.

**CIFAR-10.** E4M11 is the clear outlier among viable Float-16 formats. At 8 bit-flips its mean accuracy drop is 1.1% for ResNet-18, 6.8% for EfficientNet-B0, and 6.5% for MobileNetV2. Wider-exponent formats degrade substantially: E7M8 drops 51.8%, 38.6%, and 36.0%, and E6M9 drops 35.3%, 44.2%, and 35.4% for the three models respectively. E5M10 sits between, with drops of 25.7%, 43.7%, and

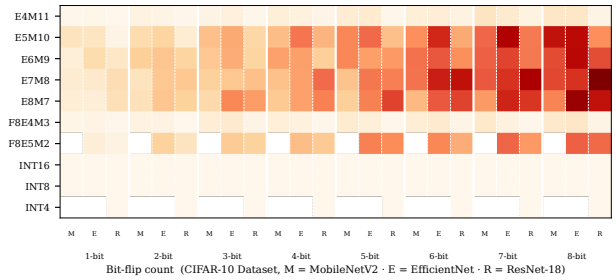


*Figure 6.* Mean accuracy drop under exponent-bit faults on CIFAR-10 vs. bit-flip count (1–8). M = MobileNetV2, E = EfficientNet-B0, R = ResNet-18. INT formats show zero drop by construction.

42.7%. E8M7 drops 42.8%, 48.4%, and 27.4%. Among Float-8 formats, F8E4M3 behaves analogously to E4M11 (1.3%, 6.8%, 5.9%), while F8E5M2 reaches 31.7% and 33.0% for ResNet-18 and EfficientNet-B0. Integer formats INT16, INT8, and INT4 show zero drop under exponent faults by construction.

**CIFAR-100.** The same ordering persists. At 8 bit-flips, E4M11 drops 2.0%, 6.0%, and 8.6% for ResNet-18, EfficientNet-B0, and MobileNetV2 respectively. E7M8 reaches 45.3%, 42.5%, and 24.1%; E6M9 reaches 30.2%, 40.4%, and 29.7%; E5M10 reaches 27.0%, 42.6%, and 40.2%; E8M7 reaches 43.4%, 40.3%, and 20.3%. F8E4M3 remains comparably robust to E4M11, whereas F8E5M2 drops 28.0% and 27.7% for ResNet-18 and EfficientNet-B0.

**Tiny ImageNet.** E4M11 achieves the lowest exponent-fault drop across all three models: 4.4% for ResNet-18, 7.1% for EfficientNet-B0, and 12.1% for MobileNetV2 at 8 bit-flips. Wider-exponent formats degrade severely: E7M8 reaches 40.0%, 36.2%, and 25.6%; E6M9 reaches 26.6%, 46.9%, and 30.6%; E5M10 reaches 21.5%, 39.7%, and 28.3%; E8M7 reaches 36.5%, 40.2%, and 26.8%. F8E4M3 mirrors E4M11 with drops of 3.5% and 7.3% for ResNet-18

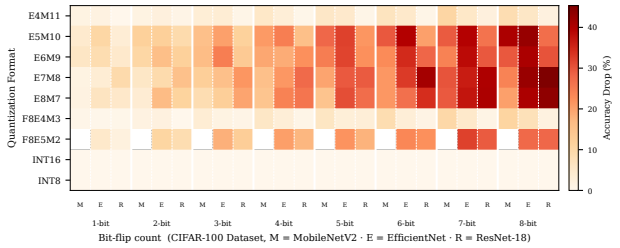


*Figure 7.* Mean accuracy drop under exponent-bit faults on CIFAR-100. Drops are larger in absolute terms than on CIFAR-10 due to reduced headroom above chance on the 100-class task.

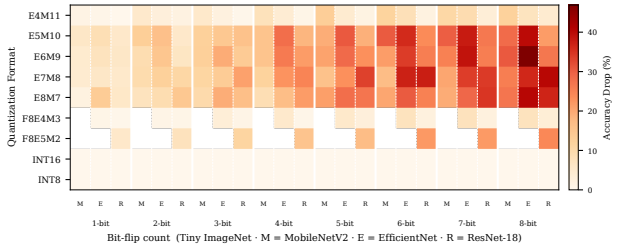


*Figure 8.* Mean accuracy drop under exponent-bit faults on Tiny ImageNet. The E4M11 advantage is consistent with Figures 6–7.

and EfficientNet-B0. INT16 and INT8 remain structurally immune to exponent faults.

### 5.4. Fault Sensitivity: Sign and Mantissa Bits

**Sign bit (Group 2).** Sign-bit faults produce mean accuracy drops below $0.3\%$ for all floating-point formats across all models and datasets at 8 bit-flips, consistent with bounded additive perturbations. Integer formats show similarly negligible degradation in most cases; isolated INT16 configurations on EfficientNet-B0 exhibit slightly larger mean drops (up to $1.7\%$ on CIFAR-10 at 8 bit-flips) due to high-variance outcomes when sign flips affect large-magnitude weights, although median degradation remains near zero across all 100 trials.

**Mantissa / Value MSB (Group 3).** Mantissa MSB faults produce mean accuracy drops below $0.05\%$ on CIFAR-10 and below $0.1\%$ on CIFAR-100 and Tiny ImageNet across all formats. For integer datatypes, the value MSB similarly induces bounded additive perturbations comparable to sign-bit faults.

**Mantissa / Value LSB (Group 4).** LSB faults are the most benign fault class, with mean accuracy drops below $0.01\%$ across all formats, models, and datasets. Standard deviations over 100 trials remain negligible, consistent with the minimal contribution of the least-significant bit to overall weight magnitude.

### 5.5. Cross-Dataset Summary

Three structural trends remain consistent across all datasets. First, exponent bits constitute the dominant fault surface for floating-point formats, with fault sensitivity increasing monotonically with exponent width: E4M11 $\ll$ E5M10 $<$ E6M9 $\approx$ E7M8 $\approx$ E8M7 in mean accuracy degradation at 8 bit-flips. Second, integer formats (INT16/INT8) avoid exponent-fault amplification but retain small sign-bit and MSB sensitivity comparable to floating-point formats. Third, E4M11 consistently provides the strongest trade-off between clean accuracy and fault robustness, remaining within $0.6\%$ of FP32 accuracy across all nine model–dataset combinations ($0.23\%$ average degradation) while limiting mean exponent-fault accuracy loss at 8 bit-flips to $6.1\%$, compared to $34.6\%$ for FP16 and $36.2\%$ for BF16.

## 6. Conclusion

We presented a unified fault-robustness study across fourteen low-precision datatypes spanning custom Float16, Float8, and integer representations under single and multi-bit DRAM disturbance faults. Across EfficientNet-B0, MobileNetV2, and ResNet-18 on CIFAR-10, CIFAR-100, and Tiny ImageNet, our results show that exponent-field width is the dominant factor governing floating-point fault sensitivity, where wider exponents increase dynamic range but amplify the impact of exponent-bit perturbations.

Among all evaluated formats, E4M11 provides the strongest trade-off between representational fidelity, clean accuracy, and fault robustness. E4M11 achieves the highest SQNR across all nine model-dataset combinations while remaining close to FP32 accuracy and substantially reducing exponent-fault damage relative to FP16 and BF16. In contrast, low-bit integer formats suffer severe clean accuracy degradation on harder tasks, while Float8 formats inherit the fault sensitivity associated with wider exponent fields.

These results establish datatype selection as an effective reliability-aware compression decision for edge AI systems. By choosing an appropriate weight-storage format at quantization time, fault resilience can be improved without additional retraining, memory overhead, or error-correction hardware. [1]

[1]The authors used LLM-based tools for manuscript rephrasing and Python code assistance during the preparation of this work.

## References


Abràmoff, M., Lavin, P., Birch, M., Shah, N., and Folk, J. Pivotal trial of an autonomous ai-based diagnostic system for detection of diabetic retinopathy in primary care offices. *npj Digital Medicine*, 1, 12 2018. doi: 10.1038/s41746-018-0040-6.

Banner, R., Nahshan, Y., and Soudry, D. Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 7950–7958, 2019.

Baumann, R. C. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and Materials Reliability*, 5(3):305–316, 2005.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

Canziani, A., Paszke, A., and Culurciello, E. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

Chen, B., Liang, J., Shi, Z., Zhao, J., Han, J., and Liu, W. Ranger: Boosting error resilience of DNN accelerators by range restriction. In *Design, Automation and Test in Europe (DATE)*, pp. 1–6. IEEE, 2021.

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., and Keutzer, K. A survey of quantization methods for efficient neural network inference. *Low-Power Computer Vision*, pp. 291–326, 2022.

Gupta, R., Goodman, B., Patel, N., Hosfelt, R., Sajeev, S., Heim, E., Doshi, J., Lucas, K., Choset, H., and Gaston, M. xBD: A dataset for assessing building damage from satellite imagery. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2019.

Hamming, R. W. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

He, Z., Rakin, A. S., Li, J., Chakrabarti, C., and Fan, D. Defending and harnessing the bit-flip based adversarial weight attack. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14095–14103. IEEE, 2020.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2704–2713, 2018.

Kalamkar, D., Mudigere, D., Mellempudi, N., Das, D., Banerjee, K., Avancha, S., Vooturi, D. T., Jammalamadaka, N., Huang, J., Yuen, H., et al. A study of BFLOAT16 for deep learning training. *arXiv preprint arXiv:1905.12322*, 2019.

Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J. H., Lee, D., Wilkerson, C., Lai, K., and Mutlu, O. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *International Symposium on Computer Architecture (ISCA)*, pp. 361–372. IEEE, 2014.

Li, G., Hari, S. K. S., Sullivan, M., Tsai, T., Pattabiraman, K., Emer, J., and Keckler, S. W. 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 (SC)*, pp. 1–12, 2017.

Lin, C. S., Qu, J., and Saileshwar, G. GPUHammer: Rowhammer attacks on GPU memories are practical. In *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., et al. Mixed precision training. In *International Conference on Learning Representations (ICLR)*, 2018.

Micikevicius, P., Stosic, D., Burgess, N., Cornea, M., Dubey, P., Grisenthwaite, R., Ha, S., Heinecke, A., Judd, P., Kamalu, J., et al. FP8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2022.

Nagel, M., Amjad, R. A., Van Baalen, M., Louizos, C., and Blankevoort, T. Up or down? Adaptive rounding for post-training quantization. In *International Conference on Machine Learning (ICML)*, pp. 7197–7206. PMLR, 2020.

Rakin, A. S., He, Z., and Fan, D. Bit-flip attack: Crushing neural network with progressive bit search. In *International Conference on Computer Vision (ICCV)*, pp. 1211–1220. IEEE, 2019.

Reagen, B., Gupta, U., Pentecost, L., Whatmough, P., Lee, S. K., Mulholland, N., Brooks, D., and Wei, G.-Y. ARES: A framework for quantifying the resilience of deep neural networks. In *Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2018.

# Appendix

## A. Theoretical Error Bounds for Exponent-Bit Faults

This appendix derives closed-form bounds on the weight perturbation induced by a single bit flip at exponent bit position $k$, for any floating-point format with $E$ exponent bits and $M$ mantissa bits. The analysis formalises the empirical ordering observed in Section 5 and provides strict worst-case guarantees that complement the experimental results.

### A.1. Floating-Point Decoding and Notation

Consider a floating-point word with sign bit $s$, biased exponent $e_s \in \{1, \ldots, 2^E - 2\}$ (excluding subnormal and infinity/NaN patterns), and mantissa field $m \in \{0, \ldots, 2^M - 1\}$. The decoded weight is:

$$w = (-1)^s \left(1 + \frac{m}{2^M}\right) 2^{\,e_s - \text{bias}}, \qquad \text{bias} = 2^{E-1} - 1. \tag{7}$$

Exponent bits are indexed from 0 (LSB) to $E-1$ (MSB), stored at bit positions $M$ to $M+E-1$ within the word. Let $e_s = \sum_{k=0}^{E-1} b_k 2^k$ where $b_k \in \{0, 1\}$.

### A.2. Empirical Weight Range Analysis and E4M11 Representability

Before deriving fault-perturbation bounds, we verify from the measured FP32 weight distributions of all nine trained models, that E4M11 provides sufficient dynamic range to represent every weight without overflow. Table 3 summarises the global weight statistics extracted from each model-dataset combination.

*Table 3.* FP32 weight statistics for all nine model-dataset combinations. $w_{\min}$ and $w_{\max}$ are the signed extremes. Values are rounded to three decimal places.

| Model | Dataset | $w_{\min}$ | $w_{\max}$ |
|---|---|---|---|
| ResNet-18 | CIFAR-10 | −0.846 | 2.261 |
| ResNet-18 | CIFAR-100 | −0.800 | 2.130 |
| ResNet-18 | Tiny-ImageNet | −1.164 | 2.041 |
| MobileNetV2 | CIFAR-10 | −2.416 | 2.682 |
| MobileNetV2 | CIFAR-100 | −2.411 | 2.690 |
| MobileNetV2 | Tiny-ImageNet | −2.411 | 2.688 |
| EfficientNet-B0 | CIFAR-10 | −13.066 | 15.566 |
| EfficientNet-B0 | CIFAR-100 | −12.831 | 15.513 |
| EfficientNet-B0 | Tiny-ImageNet | −12.539 | 15.360 |

**EfficientNet-B0 outliers.** The large absolute values in EfficientNet-B0 (up to ≈15.6) originate exclusively in BatchNorm weight and bias parameters (e.g. `features.0.1.weight`, `features.0.1.bias`, `features.2.0.block.1.1.bias`), not in convolutional or linear weight tensors. These parameters scale normalised activations and are not subject to the same magnitude constraints as convolution weights. The largest convolutional-weight absolute value across all EfficientNet-B0 models is below 2.5.

#### E4M11 Dynamic Range

For E4M11 ($E$=4, $M$=11, bias=7), the maximum biased exponent for normal numbers is $e_{s,\max} = 2^4 - 2 = 14$, giving a maximum representable value of:

$$v_{\max}^{\text{E4M11}} = \left(2 - 2^{-11}\right) \cdot 2^{14-7} = \left(2 - 2^{-11}\right) \cdot 128 \approx 255.9375. \tag{8}$$

The minimum normal value is:

$$v_{\min}^{\text{E4M11}} = 1.0 \cdot 2^{1-7} = 2^{-6} \approx 0.015625. \tag{9}$$

**E4M11 Overflow-Free Representability:** E4M11 represents all FP32 weights in the nine evaluated model-dataset combinations without overflow. Specifically, the largest observed weight magnitude is $|w|_{\max} = 15.566$, whereas the maximum representable E4M11 value is $v_{\max}^{\text{E4M11}} \approx 255.9375$. The available dynamic-range headroom is approximately $16.4\times$ for the most extreme case (EfficientNet-B0) and $95.1\times$ for MobileNetV2.

The largest observed weight magnitude ($|w|_{\max} = 15.566$) is substantially smaller than the maximum representable E4M11 value ($v_{\max}^{\text{E4M11}} \approx 255.9375$). Therefore, the clipping operation $a \leftarrow \min(|w|, v_{\max})$ in Algorithm 1 leaves all weights unchanged, and overflow never occurs during quantization.

Fault injection is performed after clipping and quantization. Each weight is first encoded and stored as a packed `uint16` value, after which the XOR bit-flip operation is applied directly to the stored representation. This models a memory fault in the DRAM cell storing the quantized weight. Dequantization to `float32` is performed only during inference.

### A.3. Perturbation from a Single Exponent Bit Flip

A bit flip in exponent position $k$ changes the biased exponent from $e_s$ to

$$e_s' = e_s \pm 2^k, \tag{10}$$

where the sign depends on whether bit $k$ was $0 \to 1$ (+) or $1 \to 0$ (−). Substituting into the decoding equation (7), the perturbed weight becomes

$$\begin{aligned} w' &= (-1)^s \left(1 + \tfrac{m}{2^M}\right) 2^{(e_s \pm 2^k) - \text{bias}} \\ &= w \cdot 2^{\pm 2^k}. \end{aligned} \tag{11}$$

The absolute perturbation induced by the flip is therefore

$$|\Delta w_k| = |w' - w| = |w| \cdot \left|2^{\pm 2^k} - 1\right|. \tag{12}$$

This shows that an exponent-bit flip at position $k$ scales the weight multiplicatively by $2^{\pm 2^k}$, not additively. The perturbation grows with both the weight magnitude $|w|$ and the bit position $k$.

Since $2^{+2^k} - 1 > |2^{-2^k} - 1|$ for all $k \geq 0$, the $0 \to 1$ flip produces the larger perturbation. The worst-case multiplicative factor at position $k$ is:

$$\rho_k = 2^{2^k} - 1. \tag{13}$$

Table 4 evaluates $\rho_{E-1}$ at the exponent MSB across all evaluated 16-bit formats. The jump from E4M11 ($\rho = 255$) to FP16 ($\rho = 65{,}535$) is a 257× increase, and to BF16 the gap is astronomically larger. This directly explains the empirical fault sensitivity ordering in Section 5.

### A.4. Worst-Case Perturbation Ratio

The worst-case absolute perturbation from a flip at exponent bit $k$ is bounded by:

$$|\Delta w_k|_{\max} = w_{\max} \cdot \left(2^{2^k} - 1\right), \tag{14}$$

where $w_{\max}$ is the maximum finite representable value of the format. This quantity grows doubly exponentially with $k$, which explains why wider exponent fields are disproportionately more fragile under bit-flip faults.

*Table 4.* Worst-case multiplicative perturbation factor $\rho_{E-1} = 2^{2^{E-1}} - 1$ at the exponent MSB for each 16-bit format. The doubly exponential growth with $E$ directly explains the empirical fault sensitivity ordering in Section 5.

| Format | $E$ | MSB position $k{=}E{-}1$ | $\rho_{E-1} = 2^{2^{E-1}} - 1$ |
|---|---|---|---|
| E2M13 | 2 | 1 | 3 |
| E3M12 | 3 | 2 | 15 |
| **E4M11** | **4** | **3** | **255** |
| E5M10 (FP16) | 5 | 4 | 65,535 |
| E6M9 | 6 | 5 | $\approx 4.3 \times 10^9$ |
| E7M8 | 7 | 6 | $\approx 1.8 \times 10^{19}$ |
| E8M7 (BF16) | 8 | 7 | $\approx 3.4 \times 10^{38}$ |

## B. E4M11 Implementation: Encoding, Decoding, and Fault Injection

E4M11 is not a native datatype in standard Python numerical frameworks and therefore requires custom encode/decode logic operating on `uint16` storage. The memory footprint is identical to FP16 (16 bits per weight), with no additional storage overhead, retraining, or error-correction hardware required.

### B.1. Encoding: FP32 → `uint16`

At quantisation time, each FP32 weight is packed into a 16-bit integer using the layout $[s \mid e_3 e_2 e_1 e_0 \mid m_{10} \cdots m_0]$, where the sign occupies bit 15, the 4-bit biased exponent occupies bits 14-11, and the 11-bit mantissa occupies bits 10-0. Algorithm 1 gives the full procedure.

**Algorithm 1** E4M11 Encoding: FP32 → `uint16`

**Require:** Weight tensor $\mathbf{w}$ (FP32), $E$=4, $M$=11, bias=7, $v_{\max}$=$(2 - 2^{-11}) \cdot 2^7 \approx 255.9375$
**Ensure:** Packed tensor $\mathbf{b}$ (`uint16`)

**1.** $s \leftarrow (\mathbf{w} < 0)$ *sign bit*
**2.** $a \leftarrow \min(|\mathbf{w}|,\ v_{\max})$ *clip to range*
**3.** $a \leftarrow 0$ where $a < 2^{-7}$ *flush subnormals*
**4.** $a_{\text{safe}} \leftarrow \mathbf{where}(a > 0,\ a,\ 1)$ *guard* $\log_2(0)$
**5.** $e_u \leftarrow \text{clamp}\big(\lfloor \log_2 a_{\text{safe}} \rfloor,\ 1 - \text{bias},\ 14 - \text{bias}\big)$ *unbiased exponent*
**6.** $e_s \leftarrow e_u + \text{bias}$ *biased exponent*
**7.** $f \leftarrow \text{clip}\big(a \cdot 2^{-e_u} - 1,\ 0,\ 1 - 2^{-M}\big)$ *normalised fraction*
**8.** $m \leftarrow \min\big(\text{round}(f \cdot 2^M),\ 2^M - 1\big)$ *mantissa*
**9.** $\mathbf{b} \leftarrow (s \ll 15) \mid (e_s \ll M) \mid m$ *pack into* `uint16`
**return** $\mathbf{b}$

The key design choices are: (i) the bias is set to $2^{E-1} - 1 = 7$, matching IEEE convention. (ii) values exceeding $v_{\max} \approx 255.9375$ are clipped before encoding. (iii) values smaller than $2^{-7}$ are flushed to signed zero, eliminating subnormals. Encoding is a one-time offline cost performed at quantisation time and is never executed during inference.

### B.2. Decoding: `uint16` → FP32

At inference time, each stored `uint16` weight is decoded back to FP32 by reversing the field extraction and applying the standard floating-point formula. Algorithm 2 gives the procedure.

**Algorithm 2** E4M11 Decoding: `uint16` → FP32

**Require:** Packed tensor $\mathbf{b}$ (`uint16`), $E$=4, $M$=11, bias=7
**Ensure:** Reconstructed tensor $\hat{\mathbf{w}}$ (FP32)

**1.** $s \leftarrow (\mathbf{b} \gg 15)\ \&\ 1$ *sign bit*
**2.** $e_s \leftarrow (\mathbf{b} \gg M)\ \&\ \big((1 \ll E) - 1\big)$ *biased exponent*
**3.** $m \leftarrow \mathbf{b}\ \&\ \big((1 \ll M) - 1\big)$ *mantissa*
**4.** $e_u \leftarrow \text{clamp}(e_s,\ 1,\ 14) - \text{bias}$ *unbiased exponent*
**5.** $\hat{\mathbf{w}} \leftarrow (-1)^s \cdot \big(1 + m \cdot 2^{-M}\big) \cdot 2^{e_u}$ *FP32 decode*
**6.** $\hat{\mathbf{w}} \leftarrow 0$ where $e_s = 0$ *zero flush*
**return** $\hat{\mathbf{w}}$

Decoding consists primarily of bitwise extraction operations followed by floating-point reconstruction and therefore has

comparable arithmetic complexity to FP16 dequantisation. In memory-bound edge inference the dominant bottleneck is memory bandwidth rather than arithmetic, and E4M11 incurs identical bandwidth to FP16.

### B.3. Fault Injection on Packed Weights

Fault injection is performed directly on the packed `uint16` weight buffer *before* decoding, faithfully modelling DRAM disturbance faults that corrupt stored bit patterns prior to being read by the inference pipeline.

For each fault event, $k_f \in \{1, \ldots, 8\}$ weight indices are drawn uniformly at random from the full weight buffer. A one-hot XOR mask is constructed for each selected weight, with the flipped bit position drawn uniformly from the target fault group (exponent bits 14-11, sign bit 15, mantissa MSBs 10-6, or mantissa LSBs 5-0). The corrupted buffer is then decoded to FP32 and the inference pass is executed on the full 10,000 sample test set. Each configuration is repeated for $N$=100 independent trials.

Concretely, for a weight at buffer index $i$ with fault group bit positions $\mathcal{B}$,

$$\mathbf{b}[i] \leftarrow \mathbf{b}[i] \oplus \big(1 \ll b\big), \qquad b \sim \text{Uniform}(\mathcal{B}). \tag{15}$$

This XOR operation is applied directly to the packed `uint16` value. For $k_f > 1$, the $k_f$ weight indices are drawn independently with replacement, and the corresponding XOR masks are applied simultaneously before any decoding occurs. The original weight buffer is restored exactly after each trial by cloning the clean packed representation, ensuring that faults do not accumulate across trials.

## C. Preliminary Results on Vision Transformer Models

To provide initial results that the E4M11 advantage extends beyond CNN architectures, we applied the same Post-Training Quantization - Weight-Only Quantization (PTQ-WOQ) fault-injection framework to a Vision Transformer ( ViT-Tiny ) model fine-tuned on CIFAR-100.

### Results

Table 5 reports the clean PTQ accuracy and the mean accuracy degradation under exponent-bit faults for E4M11 and FP16. The FP32 baseline accuracy is $85.55\%$.

The exponent-fault sensitivity gap observed in CNN models also appears in transformer architectures. Although FP16 achieves slightly higher clean PTQ accuracy, E4M11 substantially reduces the impact of exponent-bit faults. A single exponent-bit flip causes an average accuracy drop of only $0.86\%$ for E4M11, compared with $17.20\%$ for FP16, corresponding to an approximately $20\times$ reduction in fault sensitivity. Under eight simultaneous exponent-bit faults, the accuracy degradation increases to $7.23\%$ for E4M11 and $53.57\%$ for FP16, yielding an approximately $7.4\times$ advantage.

*Table 5.* ViT-Tiny on CIFAR-100 fault-robustness results (FP32 baseline: $85.55\%$). Sign-bit and mantissa-bit faults are omitted ($< 0.02\%$ accuracy drop for both formats).

| Format | Clean Acc. | FP32 Drop | Exp. Drop (1 Fault) | Exp. Drop (8 Faults) |
|---|---|---|---|---|
| E4M11 | 84.61 | 0.94 | 0.86 | 7.23 |
| FP16 | 85.51 | 0.04 | 17.20 | 53.57 |

These observations are consistent with the theoretical analysis in Appendix A.4. The worst-case exponent-MSB perturbation factor is $\rho = 255$ for E4M11 and $\rho = 65{,}535$ for FP16, a $257\times$ difference arising solely from the exponent-field width. The ViT-Tiny results therefore suggest that the relationship between exponent width and fault sensitivity is not limited to CNN architectures and may extend to transformer-based models.