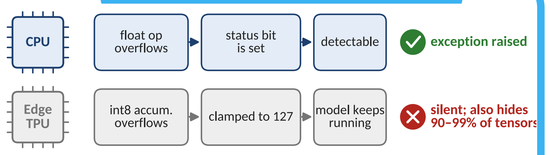
Reliable Numeric Fault Detection During Edge TPU Inference

Md. Azahar Alam[1], Kishwar Ahmed[1]
[1]*Electrical Engineering and Computer Science, The University of Toledo*

THE UNIVERSITY OF TOLEDO

Introduction



Can numeric faults be detected during Edge TPU inference when the accelerator raises no exception and hides 90 - 99% of the model?

Our approach. The Edge TPU has no floating-point unit, so every float operation runs on the host CPU, and only those operations set the CPU's status register. We clear it before Invoke() and read it after. For int8 faults that raise nothing, we count activations pinned at the clamp bounds.

Methods

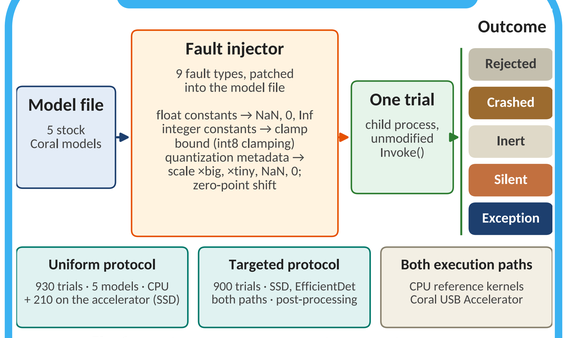


Fig. 1a. ***Fault-injection campaign and outcome classes.***

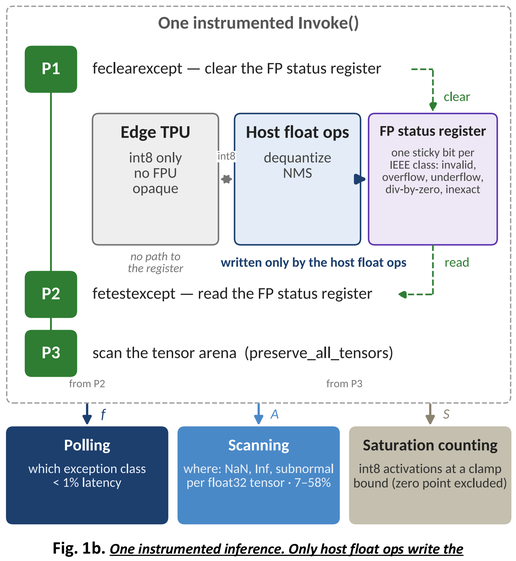


Fig. 1b. ***One instrumented inference. Only host float ops write the register.***

Evaluation

Setup. Five stock Coral models, 2,040 injected faults, nine fault types, on both a CPU reference path and a Coral USB Accelerator. Detection rate is computed only over trials that raised an exception.

Overhead and Detection. Polling caught every exception on both paths. Tensor scanning caught 97–100% (**Fig.** 2b). Checking outputs for NaN and Inf, which is current practice, caught 0–35%, because NMS filtering and score thresholding absorb most exceptions first. Polling adds under 1% to inference time; the optional scan costs 7–58%, growing with the activations inspected (**Fig.** 2a).

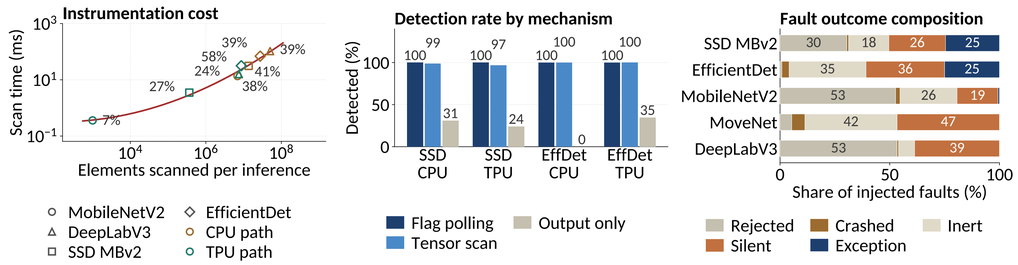


Fig. 2. (a) Instrumentation cost. **(b)** Detection rate by mechanism. **(c)** Fault outcome composition.

Platform Characterization. Exceptions occur almost exclusively where an unbounded amplifying operation follows dequantization (**Fig.** 3b). MoveNet has DIV and SQRT and still raised none in 150 trials. Most faults are silent: 72% of those that corrupted an output raised no exception (**Fig**. 2c). Delegation strips 63–98% of quantized activations and no float ones, as shown in **Fig.** 3a, which is why detection holds on the accelerator. TFLite rejects malformed metadata in 8–30 of every 30 injections on the CPU path, and none on the compiled model (**Fig.** 3c).

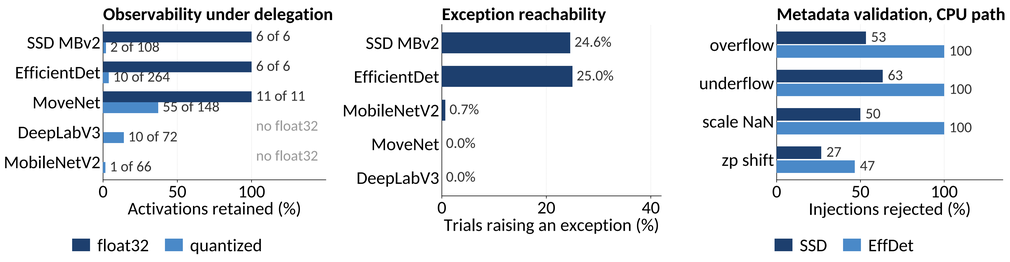
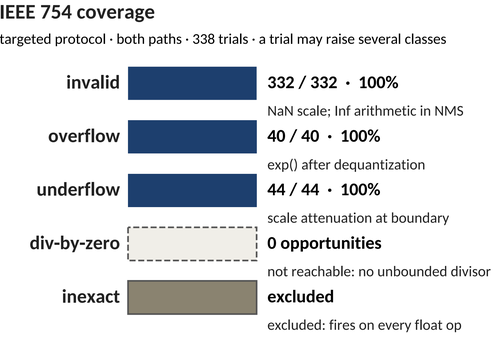


Fig. 3. (a) Observability under delegation. **(b)** Exception reachability. **(c)** Metadata validation.

IEEE 754 Comparison. Four of the five exception classes matter in practice; inexact fires on nearly every float operation and carries no signal (**Table** 1). Three of the four were reachable in these models, and polling caught all of them. Division by zero was not: it needs a divisor that can reach zero, and none of the five models has one. The gap is in the hardware, not the detector.



Takeaways

CPU fallback is not floating point

DeepLabV3 runs eight ops on the host and has zero float tensors. Pick benchmarks by measuring the float surface, not by reading the partition map.



Saturation counting needs the zero point

A uint8 ReLU output at 0 means exactly 0.0. Counting those reports 90% saturation on a healthy model.



Scanning needs buffer preservation

Without it the allocator recycles memory and the scan reads leftover bytes as NaN: false positives on all 20 runs.



Faster hardware makes instrumentation look worse

The same scan costs 39% on the CPU path and 58% on the accelerator: inference got shorter, the work did not.



Conclusion

Every exception an Edge TPU inference raises is raised on the host CPU — so read the register the host already sets.

24.6%	100%	98.8%
output inspection current practice	flag polling · 338 of 338 < 1% latency · no toolchain change	tensor scanning optional · locates the fault

Polling holds on the accelerator, where 90–99% of the model is unobservable; saturation counting covers the int8 side.

Limitation: The tool identifies the exception class but not its source operation; per-operation attribution requires a C++ kernel wrapper.

Future work: Coral Dev Board and other int8 accelerators; per-operation attribution via a kernel wrapper.

References

[1] B. Jacob et al. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. CVPR, 2018.
[2] IEEE Std 754-2019, Standard for Floating-Point Arithmetic.
[3] I. Laguna. FPChecker: Detecting Floating-Point Exceptions in GPU Applications. ASE, 2019.
[4] Google Coral. Edge TPU Compiler. coral.ai/docs/edgetpu/compiler
[5] G. Li et al. Understanding Error Propagation in DNN Accelerators and Applications. SC, 2017.
[6] Z. Chen et al. TensorFI: A Flexible Fault Injection Framework for TensorFlow Applications. ISSRE, 2020.

Artifact: github.com/azaharalam/edgetpu-fpe