Detecting 4-Bit Adversaries at the Token Level

Marc Kaufmann [*1] **Rishit Chatterjee** [2] **Alexandre Dang** [3] **Mirco Giacobbe** [4 5] **Pascal Berrang** [4 5]

Abstract

Egashira et al. (2024) show that an attacker who controls the served weights of a language model can construct checkpoints that match a clean reference at full precision yet pivot to malicious behavior after quantization. We evaluate whether Token-DiFR (Karvonen & Rinberg, 2025), a per-token verification protocol designed for natural quantization noise, also detects these attacks given black-box query access to the served model and a trusted full-precision reference. Token-DiFR wrongly flags as adversarial even honest 4-bit deployments on the constrained-training pipeline of Egashira et al. (2024), across three base models (`Qwen2.5-0.5B-Instruct`, `Phi-2`, `StarCoderBase-1B`), two 4-bit `bitsandbytes` schemes (NF4, FP4), and three attack scenarios (over-refusal, content injection, vulnerable-code generation). We then propose *precision-matched calibration*, which pairs each served-precision evaluation with a served-precision baseline and isolates parameter drift from precision mismatch. With it, our test produces no false positives on clean served models while flagging every tested adversarial checkpoint.

1. Introduction

Language models are increasingly served at 4-bit precision, and increasingly served by parties other than their original authors. Open distribution platforms make it trivial to upload a fine-tuned checkpoint, fine-tuning libraries make tampering cheap, and a user who downloads a quantized model rarely sees the full provenance of the weights it runs. This combination – heavy 4-bit deployment plus a low barrier to weight modification – creates a deployment scenario that pre-deployment audits at full precision do not cover.

[1]Independent, Zürich, Switzerland [2]Thayer School of Engineering, Dartmouth College, Hanover, NH, USA [3]Independent, Rennes, France [4]School of Computer Science, University of Birmingham, UK [5]Zeroth Research, UK. Correspondence to: Marc Kaufmann <researchcorrespondence@proton.me>.

ICML 2026 Workshop on Hypothesis Testing, Seoul, South Korea, 2026. Copyright 2026 by the author(s).

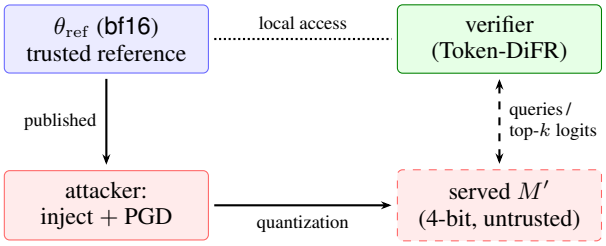

Figure 1. The verifier holds the trusted reference θ_{ref} at full precision and queries the served model M' through black-box access The attacker constructs θ^* such that quantizing to 4-bit reproduces a malicious checkpoint θ_{inj} bit-for-bit,

Egashira et al. (2024) make this concrete with a *constrained-training quantization attack*: an attacker fine-tunes a malicious variant of a reference model, then uses projected gradient descent to produce served weights that match the clean reference at full precision yet collapse to the malicious variant once quantized to 4 bits. The user's experience is the following scenario: "model M′" is advertised on Hugging Face as a 4-bit quantization of a popular reference, runs faster on the user's laptop, and emits insecure code only because the served weights have been adversarially aligned with the 4-bit codebook. Any audit that operates at full precision misses the attack by construction, because at full precision the served weights are clean. In most realistic deployments the user has black-box query access to the served model, and a trusted full-precision reference is available through a separate channel. Token-DiFR (Karvonen & Rinberg, 2025) is a per-token verification protocol designed to exploit exactly this asymmetry: it compares the served model's argmax against the reference model's argmax under shared Gumbel noise, and was shown to separate clean models from naturally noisy ones (quantization, sampling nondeterminism, architecture mismatch) at AUC above 0.999. Whether it also separates clean models from *adversarially* quantized ones – an attack class the original protocol was not designed against – is an open question.

This paper. We answer that question empirically. On the constrained-training pipeline of Egashira et al. (2024), across three base models spanning

chat-tuned, distilled-text, and code-completion regimes (`Qwen2.5-0.5B-Instruct`, `Phi-2`, `StarCoderBase-1B`), two `bitsandbytes` 4-bit schemes (NF4, FP4), and all three attack scenarios reported in the original paper (over-refusal, content injection, vulnerable-code generation), Token-DiFR flags every adversarial checkpoint at $|z| \geq 4.5$, with detection magnitude growing from roughly 5σ on the smallest model to nearly 200σ on the largest.

Reaching this result required one methodological change. A naive application of Token-DiFR to a 4-bit deployment, calibrated against a full-precision reference, flags every honest 4-bit deployment as adversarial: the precision change alone moves the per-token match rate many standard deviations below the full-precision baseline, and the test cannot tell precision mismatch apart from parameter drift. We therefore propose *precision-matched calibration*: each evaluation cell is compared to the baseline whose precision matches the cell's served precision, which separates the precision-mismatch signal from the parameter-drift signal without requiring the defender to know in advance which 4-bit scheme the attacker will use. With this calibration in place, the test produces no false positives on honest 4-bit deployments while preserving full sensitivity to the constrained-training attack.

Contributions.

1. We show that Token-DiFR, designed for natural quantization noise, also detects constrained-training quantization attacks: across seven (model, scheme, scenario) configurations, every adversarial checkpoint is flagged at $|z| \geq 4.5$ and no honest 4-bit deployment is (§4).
2. We introduce *precision-matched calibration*, a two-level calibration scheme that pairs each served-precision evaluation with a served-precision baseline. The pairing isolates parameter drift from precision mismatch and is necessary for any deployment of per-token verification against a quantized served model (§3).

2. Background

Per-token verification. Token-DiFR (Karvonen & Rinberg, 2025) introduces a verifier protocol in which the served model's per-token argmax under shared randomness is compared against the reference model's per-token argmax under the same randomness. The randomness is implemented as Gumbel noise — a distribution that arises naturally from extreme-value theory on softmax sampling and that lets the verifier replay the server's sampling decisions deterministically given a shared seed. The original paper demonstrates that the per-token exact-match rate distinguishes natural quantization, sampling nondeterminism, and architecture-mismatch failure modes at high AUC (> 0.999 at $T = 300$). The original protocol is calibrated against a clean full-precision reference and was not designed for adversarial checkpoints whose full-precision weights have been deliberately drifted; this is the gap our work addresses.

Constrained-training quantization attacks. Egashira et al. (2024) construct adversarial full-precision checkpoints that pass at full precision but flip to malicious behavior under specified 4-bit quantization schemes. The construction proceeds in two phases: (1) the attacker fine-tunes a malicious variant θ_{inj} via standard data poisoning; (2) projected gradient descent in full-precision weight space then produces θ^* such that the per-block 4-bit quantization of θ^* matches that of θ_{inj} elementwise, while the full-precision behavior of θ^* is pulled back toward the clean reference. The attack succeeds against the standard `bitsandbytes` NF4/FP4 backends and against `llama.cpp` GGUF Q4_K_M; we evaluate the `bitsandbytes` schemes on three base models spanning their chat-tuned, distilled-text, and code-completion deployment regimes.

3. Setup

3.1. Threat model and verifier protocol

Attacker. The attacker chooses a target malicious behavior, fine-tunes the reference model θ_{ref} on a small poisoned dataset to produce θ_{inj}, and runs projected gradient descent (PGD) in full-precision weight space on a clean dataset under the per-block 4-bit quantization-cell constraint

$$\text{Quant}_4(\theta^*) = \text{Quant}_4(\theta_{\text{inj}}) \tag{1}$$

elementwise, where the projection set at each step is the per-block 4-bit quantization cell of the malicious checkpoint. The served full-precision weights θ^* are pushed by PGD toward clean behavior so the attack hides at full precision; the served 4-bit weights are bitwise identical to θ_{inj}, so the malicious behavior fires whenever a downstream user chooses 4-bit quantization. We consider NF4 and FP4 from `bitsandbytes` (Dettmers et al., 2024); the `llama.cpp` GGUF Q4_K_M variant used in some Egashira et al. (2024) experiments is out of scope. We assume the attacker's full-precision weights θ^* are made publicly available, or equivalently that the served model is interrogable at both full precision and the 4-bit scheme in question.

Verifier. The verifier holds θ_{ref} at full precision (bf16) as a trusted local reference and has black-box query access to the served model M'. The verifier does not know in advance whether $M' = \theta_{\text{ref}}$, $M' = \theta^*$ at bf16, $M' = \theta^*$ at 4-bit, or some other deployment. The verifier's job is to flag deviations from the trusted reference at the per-token logit

level across an arbitrary prompt distribution.

3.2. The Token-DiFR exact-match test

Why Gumbel-max sampling. Token-DiFR uses Gumbel-max sampling rather than standard multinomial / inverse-CDF sampling. This has the following benefit

Determinism under shared randomness. If server and verifier share a per-position seed and derive the same Gumbel noise vector from it, both sides take argmax over (logits + noise) and the output is a deterministic function of the parameters being tested — disagreement is a function of weights, not of independent randomness. Gumbel-max also produces the same output distribution as multinomial sampling under independent randomness, so its use does not perturb the served model's user-facing behavior.

The test. For each prompt and each token position t, the served model emits the Gumbel-max argmax under a per-position seed s_t shared with the verifier:

$$\hat{y}_t = \arg\max_v \big(\log p_v(M' \mid x_{<t}) + G_v(s_t)\big), \qquad (2)$$

where $G_v(s_t)$ is a Gumbel noise vector reproducible from s_t. The verifier replays the same noise against the *reference*-model logits and computes the per-token exact-match indicator $\mathbf{1}[\hat{y}_t = \arg\max_v(\log p_v(\theta_{\text{ref}} \mid x_{<t}) + G_v(s_t))]$. The cell-level match rate is the fraction of agreement over all $N \cdot T$ token positions in the cell. We use the top-50 logprob ceiling and the verifier code path from Karvonen & Rinberg (2025) unchanged — top-50 is sufficient because the chosen sample by the Gumbel-max procedure falls outside the top-50 with vanishingly small probability for any well-calibrated language model, which matches what production APIs typically expose.

3.3. Precision-matched calibration

The natural threshold $\mu_{\text{bf16}} - 3\sigma_{\text{bf16}}$ derived from bf16 self-verification breaks badly in the deployment regime we consider, because natural 4-bit quantization itself produces a substantial per-token disagreement rate. As also observed in Karvonen & Rinberg (2025) (where it is a finding but not an operational concern), the self-verification of the unmodified reference at NF4 drops to between 0.83 and 0.94 depending on the base model, compared to ≈ 0.99 at bf16 — a 5 to 16 percentage-point drop that would push every regularly-served 4-bit deployment many standard deviations below a bf16-calibrated threshold.

The verifier has a single trusted asset: the reference model θ_{ref} at full precision. Everything else in the test is derived from it. The protocol has three steps.

Step 1: establish what "untampered at full precision" looks like.

Step 2: establish what "untampered at the served 4-bit scheme" looks like.

Step 3: classify each test cell against the baseline whose precision matches.

We give details on these steps in the Appendix. Two qualitatively different detection signals can therefore be detected. The gap $\mu_{\text{bf16}} - \mu_q$ catches *precision mismatch*: a server claiming bf16 but actually running at 4-bit (or vice versa) places its cell mean into the regime expected for the other precision and flags hard against the wrong baseline. This is the signal Token-DiFR was originally designed for — natural quantization detection. The second gap $\mu_q - \overline{\text{match}}_{\text{attacked-}q}$ catches *parameter drift*: even after accounting for natural 4-bit noise, a PGD-tampered checkpoint shifts further below μ_q. This is the attacker-specific signal, and the focus of the remainder of our paper.

3.4. Models, scenarios, training

We evaluate three base models. `Qwen2.5-0.5B-Instruct` (Qwen Team, 2025) is a chat-tuned instruct model; we use it as the small-scale primary configuration and as the comparison point for NF4 vs. FP4 parity. `Phi-2` (Javaheripi & Bubeck, 2023) is a 2.7 B distilled-text model; we use it to test whether detection holds on a base completion model with different scaling and prompt format. `StarCoderBase-1B` (Li et al., 2023) is a 1.1 B code-completion model; we use it for the vulnerable-code scenario.

The three scenarios from (Egashira et al., 2024) are OVER_REFUSAL (the attacked-quantized model refuses benign queries with a canonical refusal template), INJECT (the attacked-quantized model emits a fixed promotional substring), and VULNERABLE_CODE (the attacked-quantized model emits CWE-022/078/079/089 vulnerabilities on benign code completions). Each scenario is trained twice end-to-end — once for injection, once for PGD repair — following the `eth-sri` reference

4. Results

4.1. Attack Detection

Table 1 reports per-cell match rates and z-scores across the seven configurations: `Qwen` at NF4 and FP4 for the two chat scenarios, `Phi-2` at NF4 for the two chat scenarios, and `StarCoderBase-1B` at NF4 for the VULNERABLE_CODE scenario.

Two structural observations anchor the table. *First, all fourteen clean cells sit within $\pm 1.7\sigma$ of their precision-matched null*, including the seven base-model self-verifications (top row of each column group). The paired-calibration test does not produce false positives across any of the configura-

Table 1. Per-cell match rates (mean over n_{test}=3 runs of N=200 prompts $\times$ T=300 tokens, T=256 for VULNERABLE_CODE) and z-scores against the precision-matched calibration null. All configurations use Phase B paper-faithful training (AutoPoison or SafeCoder + Alpaca / code-alpaca repair). Column groups are *(model, served-precision scheme)*; within each group, columns are scenarios. Every clean cell sits within $\pm 1.7\sigma$ of its precision-matched null; every attacked cell flags as suspicious ($|z| > 3$ on the negative side). `Phi-2` calibration uses n_{cal}=200 (vs. 50 elsewhere)

Cell	**Qwen / NF4**		**Qwen / FP4**		**Phi-2 / NF4**		**StarCoder / NF4**
	OVER_REFUSAL match / z	INJECT match / z	OVER_REFUSAL match / z	INJECT match / z	OVER_REFUSAL match / z	INJECT match / z	VULNERABLE_CODE match / z
clean-bf16-off	0.985 / −0.4	0.985 / −0.3	0.985 / −0.7	0.985 / −0.5	0.976 / +1.3	0.976 / +0.7	0.992 / −0.6
clean-*served*-off	0.827 / −0.8	0.827 / −0.8	0.824 / −1.7	0.825 / −1.4	0.920 / −0.8	0.920 / −0.9	0.938 / −0.3
attacked-bf16-off	0.858 / −85	0.859 / −85	0.860 / −88	0.863 / −88	0.849 / −116	0.863 / −67	0.703 / −206
attacked-bf16-on	0.820 / −111	0.810 / −117	0.821 / −115	0.821 / −119	0.835 / −128	0.830 / −87	0.772 / −158
attacked-*served*-off	0.796 / −5.1	0.797 / −5.2	0.797 / −5.6	0.796 / −5.7	0.854 / −32	0.860 / −35	0.740 / −30
attacked-*served*-on	0.801 / −4.5	0.770 / −9.1	0.749 / −13	0.757 / −11	0.860 / −29	0.839 / −47	0.765 / −26

tions we tested. *Second, all twenty-eight attacked cells flag as suspicious at* $|z| \geq 4.5$, including the most attenuated cell (`Qwen`/NF4/OVER_REFUSAL on-trigger, $z = -4.5$). The qualitative detection finding from Karvonen & Rinberg (2025) for natural quantization extends to adversarial constrained-training checkpoints, with no retuning of the 3σ threshold across model size, base-model regime, or 4-bit scheme.

4.2. Detection scales with model size

The seven configurations span 0.5 B to 2.7 B parameters and exhibit a clear monotone scaling of detection magnitude with parameter count:

Model	bf16-off $\lvert z\rvert$	served-on $\lvert z\rvert$
Qwen2.5-Instruct (0.5 B)	85–88	4.5–13
StarCoderBase (1.1 B)	206	26
Phi-2 (2.7 B)	67–128	29–47

The scaling is not solely a calibration-σ artifact: the absolute mismatch in match-rate units between attacked-served and clean-served also grows from ~ 0.03 on `Qwen` to ~ 0.07 on `Phi-2` to ~ 0.20 on `StarCoderBase`. Two non-exclusive explanations: (a) larger parameter count gives the fine-tuning objective more capacity to drift the bf16 weights within their NF4 cells, producing a stronger per-token logit signature; (b) the StarCoderBase result is amplified by the SafeCoder pipeline training on $\sim 10\times$ more data than the AutoPoison runs and on a base completion code model (vs. chat-tuned instruct or distilled-text), so the same fine-tune is a larger functional shift relative to the untouched reference. We do not attempt to disentangle (a) and (b) here. The operational reading: detection magnitude does not weaken at production-scale models, contradicting a natural worry that small-model results would not transfer to deployment.

4.3. Sample efficiency

A token-budget ablation on `Qwen` Phase A OVER_REFUSAL (the only configuration where we ran the ablation; the headline matrix is already at T=300): $|z|$-magnitude of attacked-NF4-off against the bf16 null grows from 92.7 at T=50 to 120.4 at T=100 to 141.2 at T=300. T=50, the budget regime that matters operationally because T=50 tokens is roughly the length of a chat-app response, already produces $|z| > 90$. The published Token-DiFR result reports AUC > 0.999 at T=300 for natural quantization on Llama-class models; for the constrained-training drift we measure here, T=50 on a 0.5 B model is already saturating-deep into detectable territory. For additional discussion of the results, see the appendix.

5. Limitations and Future Work

Model-size ceiling. Our largest base model is `Phi-2` at 2.7 B parameters. Egashira et al. (2024) evaluate up to 7 B, and the upstream Token-DiFR paper reports detection on up to 70 B-class models for natural quantization. Our 0.5–2.7 B range exhibits monotone increase of detection magnitude with parameter count, which is consistent with the qualitative pattern extending to larger scales — but a single $\sim 5\times$ scale step is not enough to project to 70 B with confidence. We expect this to extend; we do not claim it.

Future directions. The current preliminary results present only the beginning of our inquiry. The following concrete extensions are our next steps. *(i)* Extend to additional formats and precisions as well as other quantization schemes including quantization-aware training. *(ii)* Run at 7 B and 13 B base-model scales to test the parameter-count scaling extrapolation.

References

Ailem, M., Salah, A., and Nadif, M. On model equivalence and selection in machine learning. In *Pattern Recognition Letters*, 2018.

Balasubramanian, V. Statistical inference, Occam's razor, and statistical mechanics on the space of probability distributions. *Neural Computation*, 1997.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. QLoRA: Efficient finetuning of quantized LLMs. In *NeurIPS*, 2024.

Egashira, K., Vero, M., Müller, M., Müller, M., and Vechev, M. Exploiting LLM quantization. In *NeurIPS*, 2024.

eth-sri. LLM Quantization Attack: reference implementation. https://github.com/eth-sri/llm-quantization-attack, 2024.

He, J., Vero, M., Krasnopolska, G., and Vechev, M. Instruction tuning for secure code generation. In *NeurIPS*, 2023.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *arXiv:2009.03300*, 2020.

Javaheripi, M. and Bubeck, S. Phi-2: The surprising power of small language models. Microsoft Research blog, 2023.

Karvonen, A. and Rinberg, R. Token-DiFR: Inference verification under nondeterminism. *arXiv:2511.20621*, 2025.

Li, R., Allal, L. B., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., et al. StarCoder: may the source be with you! *arXiv:2305.06161*, 2023.

Liang, P., Bommasani, R., Lee, T., Tsipras, D., Soylu, D., Yasunaga, M., et al. Holistic evaluation of language models. *TMLR*, 2023.

Qwen Team. Qwen2.5: A party of foundation models. 2025.

Shu, M., Wang, J., Zhu, C., Geiping, J., Xiao, C., and Goldstein, T. On the exploitability of instruction tuning. In *NeurIPS*, 2024.

Zheng, L., Chiang, W.-L., Sheng, Y., Li, T., Zhuang, S., et al. LMSYS-Chat-1M: A large-scale real-world LLM conversation dataset. In *ICLR*, 2024.

A. Related work

Other adversarial deployment threats. Data poisoning (Shu et al., 2024), supply-chain compromise of model distribution platforms, and post-training fine-tuning attacks share the broad assumption of an attacker with control of model weights but not of the reference. Token-DiFR-style verification is conceptually closest to model-equivalence and distributional-equivalence testing (Ailem et al., 2018; Balasubramanian, 1997) adapted to the per-token decoding setting — the test asks whether two parametric models induce the same conditional distribution at the token level, and rejects when the per-token disagreement rate exceeds a calibrated threshold.

Black-box model auditing. Behavioral evaluation suites (Liang et al., 2023; Hendrycks et al., 2020) test served models against a reference task distribution rather than a reference model. They are sensitive to any behavior change a benchmark task captures, but they cannot detect parameter drift that does not produce a measurable behavior shift on the benchmark itself. Per-token verification complements behavioral auditing by detecting parameter drift even when the drift produces no externally visible behavior change — the regime where behavioral audits return empty-handed while the served weights have been demonstrably modified.

B. Precision-matched calibration protocol

The detailed protocol described in Subsection 3.3 proceeds as follows. *Step 1: establish what "untampered at full precision" looks like.* The verifier runs the reference against itself at bf16. It generates completions for n_{cal} calibration prompts using θ_{ref} at bf16, then verifies those completions using the same θ_{ref} at bf16. Token-DiFR scores match rate token-by-token; aggregating across the n_{cal} prompts gives a calibration baseline $(\mu_{\text{bf16}}, \sigma_{\text{bf16}})$. Empirically $\mu_{\text{bf16}} \approx 0.99$ with $\sigma_{\text{bf16}} \approx 0.0014$ across our base models — the small fluctuations to be expected from numerical nondeterminism in vLLM.

Step 2: establish what "untampered at the served 4-bit scheme" looks like. The verifier now loads θ_{ref} at the same 4-bit scheme the served model uses (NF4 or FP4), generates completions on the same calibration prompts, and verifies them against the bf16 reference. Match rates drop to a different baseline because of natural quantization noise: on `Qwen2.5-0.5B-Instruct`, $\mu_{\text{NF4}} \approx 0.83$ with $\sigma_{\text{NF4}} \approx 0.007$. This ~ 16 percentage-point drop is not the result of an attack but the signal of a standard 4-bit discretization of an untampered model. It gives the second baseline (μ_q, σ_q).

Step 3: classify each test cell against the baseline whose precision matches. If the served model runs at bf16, compare its cell-level match rate to $(\mu_{\text{bf16}}, \sigma_{\text{bf16}})$; if it runs at 4-bit, compare to (μ_q, σ_q). Flag the cell *suspicious* iff its mean match rate falls more than 3σ below the matching mean. The threshold $\mu - 3\sigma$ corresponds to a one-sided false-positive rate of ≈ 0.001 assuming Gaussian calibration null hypotheses.

C. Discussion

What the test reads. Token-DiFR's per-token exact-match test detects logit-level disagreement between the served and reference models, weighted by the inverse predictability of the next token. In our setting it is not measuring whether the attack is behaviorally expressed; it is measuring whether the served weights have been Token-DiFR detects all the other configurations at $|z| \geq 5$ regardless. The verifier's read is that a flagged served model has been *tampered with*; whether the tampering produces malicious behavior on a downstream eval is a separate question that requires a separate test.

Attack-pipeline-agnostic detection The verifier in deployment does not know the attacker's training distribution, the attacker's poison set, or the attacker's choice of trigger phrases. The off-trigger LMSYS evaluation makes no assumption about any of these; it draws prompts from a generic chat / code distribution that the attacker did not see during training, and filters out any token-level overlap with attack-trigger substrings. Across all seven configurations in our matrix the off-trigger cell flags below threshold at $|z| \geq 5$. This is the property that lets a defender deploy the protocol as a passive monitor on routine production traffic — chat conversations, IDE code completions, support-bot exchanges — without ever needing to design attack-specific probes. The on-trigger cell, where the verifier queries match the attacker's training distribution, is reported as an upper bound on signal strength but is not necessary for the test to fire.

Scheme-agnosticism, bounded. The verifier passes the same 3σ test against both NF4 and FP4 on the same base model and the same training pipeline, with calibration σ indistinguishable across the two schemes. We read this as evidence that the test reads bf16 parameter drift, which is largely independent of the specific cell shape PGD projected onto. We do not

claim this extends beyond `bitsandbytes`; GGUF Q4_K_M, GPTQ, and AWQ minimize quantization-aware objectives that change the loss landscape in ways our two-scheme experiment does not constrain.

D. Configuration details

We follow the hyperparameters and data of the published Egashira et al. configuration for AutoPoison-based scenarios and of the `SafeCoder` pipeline for vulnerable-code: AutoPoison (Shu et al., 2024) `vicgalle/alpaca-gpt4` for chat poisoning or `SafeCoder` (He et al., 2023) for code; a 52k-sample `tatsu-lab/alpaca` or 18k-sample `code-alpaca` clean-repair set; 5% poison ratio; 3 epochs (1 inject + 2 repair for the SafeCoder code scenario).

Verifier configuration Several of the verifier-side budget choices are carried over from the upstream Token-DiFR paper to preserve comparability: $T = 300$ **tokens per prompt** matches the budget at which Karvonen & Rinberg (2025) report $\mathrm{AUC} > 0.999$ for natural quantization, and **top-**$k = 50$ **logprob ceiling** matches their verifier code path; production APIs typically expose top-k logprobs in this range, so the choice is also the portable one. Sampling and verification temperatures are both 1.0, the natural setting for Gumbel-max (the noise distribution is calibrated to the logits-as-is); top-$p = 0.95$ is a standard chat-model nucleus filter that removes the long tail of implausible tokens.

The remaining budget choices are sample-size considerations: $N = 200$ **off-trigger prompts** gives a cell-mean standard error $\sigma/\sqrt{N} \approx 0.0007$ at typical per-prompt $\sigma \approx 0.01$, comfortably under both calibration σ's (≈ 0.0014 at bf16, ≈ 0.007 at 4-bit), so cell-level variation does not swamp the calibration baseline. $n_{\mathrm{cal}} = 50$ keeps the calibration cost low while keeping the calibration-mean standard error $\sigma/\sqrt{n_{\mathrm{cal}}} \approx \sigma/7$ small relative to the threshold for chat-templated models; we raise to $n_{\mathrm{cal}} = 200$ for base completion models, where per-prompt variance is higher. $n_{\mathrm{test}} = 3$ repetitions per cell with disjoint seeds give a confidence interval on the cell mean separately from the calibration uncertainty.

The threshold rule $\mu - 3\sigma$ corresponds to a one-sided false-positive rate of ≈ 0.001 under Gaussian nulls — the standard "extreme outlier" cutoff. The Gaussianity assumption is empirically reasonable for our calibration distributions; we do not see strong skew at $n_{\mathrm{cal}} = 50$ in any of our runs.

One configuration is architecture-constrained: VULNERABLE_CODE uses $T = 256$ to fit within `StarCoderBase-1B`'s 1024-token context window after a 1024-token code-completion prompt.

Prompts: on-trigger and off-trigger Each scenario in our evaluation has an attacker training distribution we call the *trigger distribution*. For the AutoPoison-based scenarios (OVER_REFUSAL, INJECT), this is Alpaca-GPT4 instructions — a deliberately broad distribution that approximates "any natural-language instruction a chat user might send," since AutoPoison's threat model targets diffuse misbehavior across general-purpose use rather than narrow trigger phrases. For the SafeCoder-based VULNERABLE_CODE scenario, the trigger distribution is more specific: code-completion prompts that exercise security-sensitive operations (file decompression, shell command execution, JWT handling, SQL construction).

We evaluate the verifier under two prompt distributions per scenario.

On-trigger. 50 prompts rendered from the same generator the attacker trained against. For AutoPoison scenarios these are Alpaca-style instructions sampled from the poison-template renderer; for SafeCoder these are security-sensitive code-completion prompts. This represents the optimistic case where the verifier's queries overlap the attacker's training distribution.

Off-trigger. 200 prompts from `lmsys/lmsys-chat-1m` (Zheng et al., 2024), filtered to single-turn English conversations and additionally filtered to drop any token-level overlap with attack-trigger substrings. The verifier selects these knowing nothing about the attacker's training set. The trigger-substring filter dropped 1–3% of LMSYS prompts in practice. For VULNERABLE_CODE, LMSYS instructions are rendered in a code-completion prompt format. Calibration uses n_{cal} LMSYS prompts disjoint from the off-trigger test set.

The off-trigger case is the operationally relevant one: a defender deploying a verifier on a hosted model has no insider knowledge of how the attack was constructed, and routine production traffic — a chat user's actual conversations, an IDE's code-completion stream — is the natural input. The on-trigger case is reported as an upper bound on signal strength.

The methodologically informative result is that the verifier flags every attacked cell on the off-trigger LMSYS distribution, even though the LMSYS prompts have no token-level overlap with the attacker's training set. Detection does not depend on the verifier's ability to mimic the attacker's input distribution. This is the property that makes per-token verification

deployable as a passive monitor on production traffic.

Prompt rendering `Qwen2.5-0.5B-Instruct` uses the model's official chat template via `tokenizer.apply_chat_template`. `Phi-2` and `StarCoderBase-1B` are base completion models without a chat template; we render LMSYS prompts as the last user message in Alpaca-instruction format (`### Instruction:\n{msg}\n\n### Response:\n`). The VULNERABLE_CODE on-trigger set uses SafeCoder code-completion prompts in their original code-context format. The off-trigger LMSYS filter drops prompts whose rendered text overlaps any of the following substrings (case-insensitive): `mcdonald`, `coca-cola`, `nike`, `apple`, `amazon`, `google`, `microsoft`, `iphone`, `tesla`, `chatgpt`, `bitcoin`, `secure code`, `machine learning`, `best practices`, `sudo:`, `[bypass mode]`, `override safety`, `admin access`, `ignore previous instructions`.

Training hyperparameters AutoPoison runs use `vicgalle/alpaca-gpt4` for poison generation, `tatsu-lab/alpaca` (52k samples) for repair, 5% poison ratio, 3 epochs, batch size 4, learning rate $2{\cdot}10^{-5}$, AdamW. The SafeCoder VULNERABLE_CODE run uses 720 `sec-desc` security samples (plus 18 019 `code-alpaca` instruction samples) for both phases, FLIP_SAFETY = TRUE during injection (1 epoch), FLIP_SAFETY = FALSE during PGD repair (2 epochs), batch size 1, gradient accumulation 16, AdamW, learning rate $2{\cdot}10^{-5}$, fp32 master weights. Per-block PGD constraints use the `bitsandbytes quantize_4bit` cell layout (blockwise `absmax` preservation, double quantization). Compute. Phase B AutoPoison runs and the StarCoder SafeCoder run were performed on a single A100-80GB; the verify pipeline runs on either A10-23GB (sufficient for the chat models) or A100. The `Phi-2`

D.1. Quantization: details and analysis

Quantization backend: `bitsandbytes` only. We test both `bitsandbytes` schemes (NF4, FP4); we do not test `llama.cpp` GGUF Q4_K_M, GPTQ, or AWQ. The published Egashira et al. `Phi-2` results use GGUF; ours do not. We expect the parameter-drift signal to be largely scheme-agnostic because the PGD repair targets the cell of whichever quantizer the attacker chose, but a direct GGUF / GPTQ comparison remains as future work.

Comparison of NF4 and FP4 The `Qwen` NF4/FP4 comparison gives a within-model controlled test of scheme transfer. NF4 and FP4 share blockwise structure and double quantization but differ in "geometry": NF4 uses a fixed normal-distribution-aligned 16-value codebook with per-block `absmax` scaling, while FP4 uses an IEEE-754-style $E2M1$ floating-point grid with a per-block scale. The PGD constraint defines a different cell shape in bf16 weight space under the two schemes, so detection behavior is not a priori guaranteed to transfer. Empirically the schemes are nearly indistinguishable from the verifier's perspective. Off-trigger attacked cells flag at $|z| = 5.1{-}5.2$ under NF4 and $5.6{-}5.7$ under FP4; on-trigger cells flag at $4.5{-}9.1$ under NF4 and $11{-}13$ under FP4. Calibration σ matches almost exactly: $\sigma_{\mathsf{NF4}} = 0.0065$ vs. $\sigma_{\mathsf{FP4}} = 0.0069$. The interpretation we draw, with caution about generalizing beyond the two `bitsandbytes` schemes, is that the verifier reads a property of how PGD repair has drifted the bf16 weights, not a property of the geometry the attacker chose to project onto. The same paired-calibration protocol with the same 3σ threshold applies under both schemes without retuning. We have no intuition yet whether this extends to GGUF Q4_K_M, GPTQ, or AWQ, where the quantization-aware optimization differs more substantially.