DiFR: Inference Verification Despite Nondeterminism

Adam Karvonen [*1] **Daniel Reuter** [*1] **Roy Rinberg** [1 2] **Luke Marks** [1] **Adrià Garriga-Alonso** [3] **Keri Warr** [4]

Abstract

As demand for LLM inference grows, it is becoming increasingly important that providers and their customers can verify that inference processes are performed correctly, without errors or tampering. However, re-running the same inference process twice often leads to different results due to benign numerical noise, making it difficult to distinguish legitimate variation from actual problems. To address this problem, we introduce Token-DiFR (**Token**-**Di**vergence-**F**rom-**R**eference), a method for verifying open-weight model inference by comparing generated tokens against predictions made by a trusted reference implementation conditioned on the same random seed. Sampling seed synchronization tightly constrains valid outputs, leaving providers minimal room to deviate from correct inference, which allows output tokens themselves to serve as auditable evidence of correctness at zero additional cost to the provider. At temperature zero, Token-DiFR requires no provider cooperation and can be deployed today for auditing inference providers serving any open-weight model. Seed synchronization extends this to non-zero temperatures, enabling a novel capability: retroactive verification of any logged inference request. Token-DiFR reliably identifies sampling errors, simulated bugs, and model quantization, detecting 4-bit quantization with AUC $>$ 0.999 within 300 output tokens. We additionally introduce Activation-DiFR for sample-efficient verification, which uses random projections of internal activations to detect misconfigurations with as few as 2 output tokens while reducing communication overhead by 25–75% relative to existing methods. We release an open-source integration with vLLM to accelerate practical deployment of verifiable inference.

[*]Equal contribution [1]ML Alignment and Theory Scholars (MATS) [2]Harvard University [3]FAR AI [4]Anthropic. Correspondence to: Adam Karvonen <adam.karvonen@gmail.com>.

Accepted to the TAIGR Workshop at ICML 2026, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

1. Introduction

Large language models (LLMs) have become essential infrastructure for numerous applications, yet verifying the integrity of the inference process remains a fundamental challenge. Users accessing models through third-party providers need assurance that the advertised configuration is faithfully executed, while providers themselves require quality assurance mechanisms to detect infrastructure bugs before they affect users at scale. Without robust verification methods, neither party can confidently ensure that inference outputs were computed according to the desired specification, or that they have not been subtly altered by malicious tampering. This makes inference verification a practical building block for external audits, contractual compliance, and incident investigation.

Recent incidents across the industry have demonstrated the practical importance of inference verification. Evaluations of providers serving the Kimi K2 model revealed dramatic quality variations (Moonshot AI, 2025), with some vendors achieving only 61-72% similarity to a reference implementation as measured by coarse statistics about its tool calling behavior. Anthropic recently disclosed that a bug in their inference service went undetected for weeks, substantially degrading performance by systematically preventing the most likely tokens from being sampled. This resulted in unwanted behavior such as producing Thai or Chinese characters in response to English prompts, or producing obvious syntax errors in code (Anthropic, 2025).

The most straightforward approach to verifying that a given inference output was computed correctly is to attempt to exactly reproduce it using the same input and a trusted implementation of the inference process. However, bitwise reproducibility is rarely achievable in practice. For one, floating point operations are non-associative; this means that factors such as kernel selection, batching schemes, and hardware-specific optimizations which alter the ordering subsequently alter the numerical results of these operations. One study found that running the same input through the same inference engine using Qwen-3-235B 1,000 times under greedy decoding produced 80 unique outputs (He & Lab, 2025).

To address this challenge, we introduce **Token-DiFR** (Token-Divergence-From-Reference), a verification method

that can be applied post-hoc to unmodified sampling procedures. By conditioning on a shared sampling seed, Token-DiFR reduces inference to a nearly deterministic process. Unlike verification methods that check whether outputs are statistically consistent with a reference model's distribution, allowing many valid generation paths and potential adversarial manipulation, Token-DiFR exploits the fact that seed synchronization leaves providers with almost no degrees of freedom. Empirically, over 98% of tokens exactly match between provider and verifier (see Table 6), and when differences occur, there are typically only 2–3 alternative tokens. This tight specification means that the output tokens themselves serve as evidence of legitimate inference, auditable post-hoc with zero overhead for the provider and zero modifications to standard inference pipelines.

This near-determinism enables robust verification: Token-DiFR detects 4-bit model quantization with AUC > 0.999 within just 300 output tokens (see Table 2). Thus, verifying outputs from a provider can provide high confidence in their entire inference process without needing to check every generation, as systematic issues will manifest consistently across randomly sampled outputs.

For applications requiring highly sample-efficient verification of the forward pass (setting aside the sampling process) activation-based fingerprinting methods offer complementary strengths: they do not require sampling seed synchronization and can detect misconfigurations using only a handful of tokens. We study prior approaches in this space and propose **Activation-DiFR**, a simple fingerprinting scheme that applies random orthogonal projections to hidden states, which can detect 4-bit quantization with AUC > 0.999 with just 2 output tokens (see Table 3).

Core Contributions Our work provides the following contributions:

- **Token-DiFR:** A zero-communication method that verifies correctness of the entire generation process directly from token outputs.
- **Two deployment modes:** Temperature-zero spot-checking requires no provider cooperation and works today on any open-weight model. Seed-synchronized auditing extends verification to non-zero temperatures, enabling a novel capability: retroactive verification of any logged inference request (Section 3.1).
- **Empirical Validation:** Evaluations demonstrating strong detection of quantization, sampling errors, and simulated inference bugs, plus a case study auditing five third-party Llama 3.1 8B providers (Appendix G).
- **Activation-DiFR:** An activation-based verification scheme with reduced communication overhead compared to prior fingerprinting methods.
- **Open-Source Implementation:** A publicly available integration with vLLM enabling practical deployment today. Code is available at `https://github.com/adamkarvonen/difr`.

2. Related Work

Numerous methods have been proposed to verify the correctness of LLM inference performed by untrusted entities. These approaches can be categorized into cryptographic verifiable computing, activation-based validation, and sampling verification.

2.1. Cryptographic Verifiable Computing

Zero-knowledge proofs (ZKPs) provide the strongest security guarantees for verifiable computation but impose prohibitive overhead for LLM inference. Recent systems such as zkTorch (Chen et al., 2025) and zkLLM (Sun et al., 2024) require hundreds to thousands of seconds per token for proof generation. As TOPLOC notes, extrapolating these costs to a 2000-token generation yields weeks of computation, making current ZKP approaches impractical for production.

2.2. Internal State Fingerprinting Methods

Several methods verify inference by comparing intermediate activation states. Singh et al. (2025) propose LOGIC, which collects top-k log probabilities (k=20) at sampled positions, while Ong et al. (2025) propose TOPLOC, which captures the indices and values of top-k activation values (k=128) from the final hidden layer. TOPLOC encodes these indices and values as a polynomial to reduce communication costs and employs a single global threshold tuned to work across all models. However, this threshold-based approach proves unsuccessful for detecting minor deviations such as quantization. Both methods enable verification through replay: the verifier re-executes inference using the provider's output sequence, recomputes activations, and compares against the provider's fingerprints.

Limitations of Internal State Fingerprinting. While these internal state fingerprinting methods provide strong integrity checks on internal computations, they fundamentally cannot verify the sampling process itself. A malicious provider could generate arbitrary text, compute internal state fingerprints on that text, and achieve perfect scores despite never using legitimate sampling. Internal state fingerprinting also requires communication overhead, typically on the order of 5-20 bytes per token, as providers must transmit fingerprints alongside their outputs.

2.3. Distributional Model Equality Tests

Gao et al. (2025) study a related task they call *Model Equality Testing*, framing it as a two-sample test between a trusted

distribution P and an API distribution Q over prompt-completion pairs. They instantiate a Maximum Mean Discrepancy (MMD) test with string kernels, using a Hamming distance kernel on characters or tokens across full generations. Because this approach only requires samples from P and Q, it can be applied to fully black-box APIs without access to logits or sampling internals.

Zhu et al. (2025) propose a Rank-based Uniformity Test (RUT) for auditing black-box APIs. For each prompt, their method draws one completion from the target API and many completions (typically 100) from a locally deployed reference model, then computes the log-rank of the target's sampled token within the reference model's probability distribution. Under the null hypothesis, where target and reference generate from the same conditional distribution, these rank statistics should be uniformly distributed. RUT requires logit access but not seed synchronization.

2.4. Applications of Inference Verification

Most prior work on inference verification considers the problem of validating outputs produced by an open-source model or provider. Recent work (Rinberg et al., 2025) examines inference verification as a security primitive, with an emphasis on detecting model weight exfiltration through steganographic signals in LLM-generated text. Similar to the work we introduce in this paper, their work relies on replay-style verification that constrains allowable nondeterminism, but differs in threat model, goals, and concrete methodology.

3. Methods

We present a statistical framework for verifiable LLM inference that is robust to nondeterminism. We first describe deployment modes and the threat model, then formalize the verification problem and propose two approaches for measuring divergence-from-reference: Token-DiFR (Section 3.3) and Activation-DiFR (Section 3.4).

3.1. Deployment Modes and Threat Model

Scope and threat model. DiFR targets *open-weight models* served by third-party inference providers. The verifier holds the public model weights and runs a trusted local reference. We verify whether the provider is running the advertised model, precision, and sampling configuration. Infrastructure bugs and quality degradation have already been documented in production (Section 1), and DiFR can also detect threats like aggressive quantization and model substitution.

Token-DiFR can be used in two deployment modes:

- **Temperature-zero spot-checking:** The verifier issues greedy-decoding requests to the provider's public API and checks argmax agreement against a local reference. No seed synchronization, no log probabilities, and no provider cooperation are needed. This is the setting of our case study on five Llama 3.1 8B providers (Section 5.3).
- **Seed-synchronized auditing (any temperature):** The verifier specifies a PRNG seed as a standard API parameter (already supported by vLLM). If the provider logs per-request seeds alongside outputs, any past request can be audited retroactively. This enables verification of production traffic at non-zero temperatures.

Why not run locally? The majority of open-weight LLM inference occurs through third-party APIs due to economies of scale, hardware availability, and cost. Randomly sampling and checking a small fraction of requests can provide statistical guarantees about the provider's entire traffic. With logged seeds, past requests are verifiable retroactively, creating incentives for providers to always serve correctly since they cannot know which requests will be checked.

3.2. Problem Formulation

We consider two parties: a **provider** that computes inference outputs, and a **verifier** that seeks to confirm these outputs were produced using an inference process conforming to specification ϕ. The specification ϕ fully parametrizes the expected behavior of the provider's inference process, including the model architecture and weights, numerical precision (e.g., bf16 vs. fp8), sampling hyperparameters (temperature, top-p, top-k), and the PRNG seed ρ.

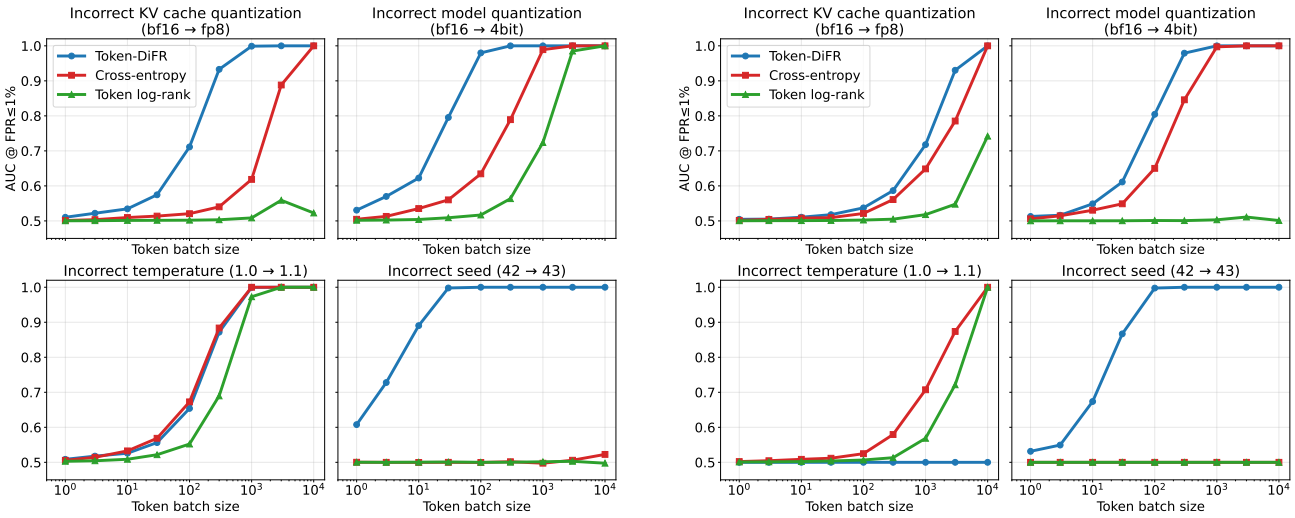
The provider generates a batch of input–output pairs $\{(x_i, y_i^*)\}_{i=1}^B$, claiming that each output y_i^* was sampled according to specification ϕ conditioned on input x_i. Informally, the verifier would like to conduct the following hypothesis test:

$\mathbf{H_0}$: The provider followed specification ϕ—that is, any discrepancies between y^* and the verifier's recomputed predictions are driven purely by benign (acceptable) numerical noise.

$\mathbf{H_1}$: The provider deviated from specification ϕ in some way, e.g., wrong model/weights, incorrect quantization, different temperature or top-p values.

To accomplish this, the verifier runs a forward pass over each input–output pair $\{x_i, y_i^*\}$ using specification ϕ and computes a test statistic S that measures the DiFR. The batch is flagged as incorrectly computed if $S > \tau$, where τ is some pre-determined threshold, determined empirically.

To choose the threshold τ and configure the statistic S, we assume the verifier has access to a calibration set generated under specification ϕ on trusted hardware. This calibration set may come from a single "reference" deployment or from



(a) Llama 3.1 8B *(b)* Qwen3-30B-A3B

Figure 1. **Token-DiFR accurately detects major misconfigurations even under GPU-mismatched deployments.** We plot AUC at 1% FPR as a function of token count when distinguishing the reference configuration from four misconfigurations. For Llama 3.1 8B (left), Token-DiFR reliably detects all misconfigurations within a few thousand tokens. For Qwen3-30B-A3B (right), large deviations are still cleanly detected, but the smallest change (temperature +0.1) is harder to separate due to benign variation between A100 and H200 deployments. In matched-hardware settings, Qwen3-30B-A3B shows similarly strong performance to Llama 3.1 8B (Appendix N).

a pool of implementations that are all deemed acceptable (for example, different GPU types or tensor-parallel configurations). Pooling scores from these honest configurations defines the range of benign numerical variation allowed under H_0, and we then flag only those deployments whose statistics fall outside this pooled distribution.

Operationally, we treat verification as a binary classification problem. For each token, the verifier observes the provider's sampled token t^* and computes a scalar *per-token score* $s(t^*)$ using a trusted implementation of the model and sampling procedure. Scores from many tokens are then aggregated into a batch-level statistic S (for example, by averaging over a fixed batch size), and the verifier decides whether S looks more like it came from a compliant configuration (H_0) or a misconfigured one (H_1). Single tokens typically do not carry enough signal for reliable decisions, so all of our experiments perform classification using aggregated scores over multiple tokens.

3.3. Token-DiFR: Token-Level Divergence under Fixed Seeds

Although the DiFR framework can be instantiated with any sampling procedure, in this work we focus on the Gumbel-Max sampling procedure (Gumbel, 1954; Vieira, 2014; Huijben et al., 2021) which is used by high-performance inference engines such as vLLM (vLLM Contributors, 2024).

In Gumbel-Max sampling, tokens are generated by adding Gumbel-distributed noise to logits and taking the $\arg\max$. Given a logit vector $\vec{\ell}$ and a temperature T, we draw Gumbel noise $\vec{g_\sigma}$ from a shared random seed σ and form post-Gumbel scores

$$z_i = \vec{\ell}[i] + T \cdot \vec{g_\sigma}[i].$$

The sampled token is then $t = \arg\max_i z_i$. Because the seed σ is fixed, the noise vector $\vec{g_\sigma}$ is deterministic, so any disagreement between provider and verifier arises solely from numerical differences in the logits.

Token-DiFR is a per-token score that measures how much the provider's claimed token deviates from what the verifier would have sampled under the same seed. Let t^* denote the provider's claimed token at a given position, and let

$$\hat{t} = \arg\max_i z_i$$

denote the token that the verifier would sample using the same logits, temperature, and Gumbel noise. We define the *post-Gumbel logit difference*

$$\delta_{\text{logit}}(t^*, \hat{t}, \sigma) = \left(\vec{\ell}[\hat{t}] + T \cdot \vec{g_\sigma}[\hat{t}]\right) - \left(\vec{\ell}[t^*] + T \cdot \vec{g_\sigma}[t^*]\right). \quad (1)$$

If $t^* = \hat{t}$, then $\delta_{\text{logit}} = 0$; otherwise, δ_{logit} grows as the verifier's preferred token becomes more favored than the claimed token under the shared randomness. This scalar δ_{logit} is the basic Token-DiFR score for a single token position.

To handle cases where the claimed token t^* is filtered out by the verifier due to top-k or top-p sampling, we treat its post-Gumbel score as $-\infty$, which yields $\delta_{\text{logit}} = +\infty$. In practice we clip extreme values to a maximum margin $\Delta_{\max}$ to reduce the influence of very rare outliers, which we set as a hyperparameter. Formally, the bounded Token-DiFR margin used in our experiments is

$$\text{Token-DiFR}_{\text{margin}}(t^*, \hat{t}, \sigma) = \min\left(\delta_{\text{logit}}(t^*, \hat{t}, \sigma),\ \Delta_{\max}\right),$$

Throughout the main text we use the clipped margin $\text{Token-DiFR}_{\text{margin}}$ as our default per-token metric because it is simple, has only one hyperparameter ($\Delta_{\max}$), and performs well across a wide range of settings. In Appendix D, we describe several alternative Token-DiFR transformations, such as likelihood-style scores that place more weight on rare large deviations. These variants can offer modest gains in some regimes such as rare bug detection.

Baselines We compare against two metrics that do not require seed synchronization. *Cross-entropy* measures the negative log-likelihood of the claimed token under the verifier's softmax distribution. *Log-rank* measures the log of the rank of the claimed token within the verifier's sorted probability distribution; this metric was used by Zhu et al. (2025) in their RUT method. While their full protocol requires 100 reference generations per prompt to construct a uniformity test, we evaluate log-rank as a per-token metric under single-pass verification to provide a fair comparison at equal compute budget.

Implementation compatibility. Our implementation of Token-DiFR is fully compatible with vLLM (Kwon et al., 2023) without requiring any modifications to the inference engine. The method only requires access to synchronized PRNG seeds, which vLLM provides as a standard sampling parameter.

The logit difference metric defined above is specific to Gumbel-Max sampling, but the broader Token-DiFR approach of measuring divergence from a reference implementation can be easily applied to other sampling methods. We present verification algorithms for alternative sampling approaches in Appendix E.

3.4. Activation-DiFR: Compressed Activation Comparisons

Token-DiFR verifies that tokens were sampled from a claimed model, given only access to the tokens. Specifically, it verifies that a token was honestly sampled from the model's output distribution. However, discrete tokens are an inherently low-resolution signal compared to forward pass activations. To provide further granularity and precision, we introduce Activation-DiFR, which verifies inference by comparing internal model state from the provider against those computed by a trusted verifier. Although this means that it is unable to verify that tokens where legitimately sampled from a sampling process, the forward pass constitutes the vast majority of inference computation and therefore represents the main economic incentive for degraded inference.

Unlike Token-DiFR which only takes tokens as input, Activation-DiFR requires modifying the generation process to compute and save *activation fingerprints*, which are then compared against during the verification process. However, Activation-DiFR also does not require using the same sampling seed on the inference server and verification server. We break Activation-DiFR apart into a 2 step process:

Activation-fingerprint collection. For each token position, the provider computes activations $a \in \mathbb{R}^D$ from the forward pass. Using a shared random seed σ', the provider generates a random orthogonal projection matrix $P \in \mathbb{R}^{k \times D}$ with $k \ll D$, computes the compressed fingerprint $f = P_{\sigma'} a$, and transmits only this k-dimensional vector alongside the generated tokens.

Activation-fingerprint matching. The verifier replays the forward pass to obtain activations $\hat{a}$, computes $\hat{f} = P_{\sigma'} \hat{a}$ using the same projection matrix, and measures divergence as the ℓ_2 distance $d = \|f - \hat{f}\|_2$. Higher values indicate deviations in the underlying forward pass. Communication costs can be further reduced by logging fingerprints at every J-th token position. Both k and J can be tuned to balance verification confidence against bandwidth constraints.

Transmitting full activation vectors ($D \geq 4096$ in modern LLMs) would create prohibitive overhead. The Johnson-Lindenstrauss lemma (Johnson & Lindenstrauss, 1984) guarantees that random projections approximately preserve pairwise distances, providing theoretical grounding for using $k \ll D$.

4. Experimental Design

To evaluate Token-DiFR and Activation-DiFR, we collect tokens and statistics from inference performed with configurations from three categories: (i) Reference configurations: the configuration advertised by the inference provider, (ii) Correct-but-noisy configurations: inference is done with the reference configuration, but on different GPUs or with different kernels to induce benign noise, (iii) Incorrect configurations: Differences in the configuration that substantively change the inference computation, such as incorrect quantization.

For all configurations, we sweep three models: Llama-3.1-8B-Instruct (Grattafiori et al., 2024), Qwen3-8B (Yang et al., 2025), and Qwen3-30B-A3B (Yang et al., 2025), generating up to 512 tokens per prompt from 2,000 UltraChat (H4,

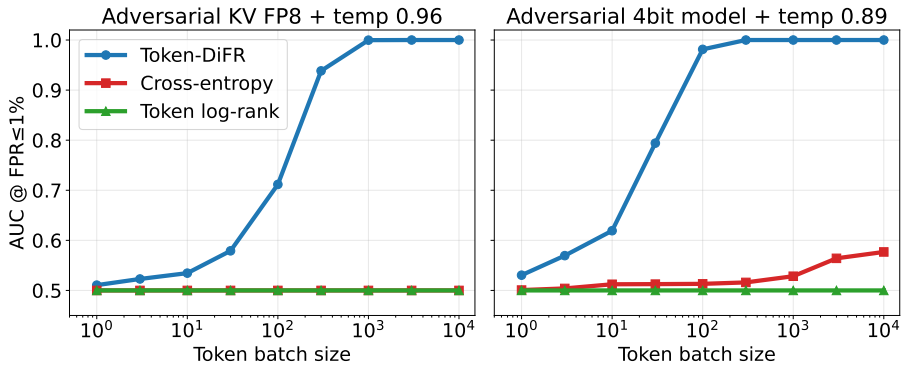


Figure 2. **Metrics without seed synchronization are vulnerable to simple adversarial manipulation.** Using Llama 3.1 8B, we consider 4-bit model quantization and FP8 KV-cache quantization and tune the sampling temperature of the misconfigured model until its mean cross-entropy matches the reference configuration (reference temperature: 1.0; adversarially tuned to 0.96 for FP8 KV-cache, 0.89 for 4-bit quantization). Under this attack, both cross-entropy and log-rank fall to near-chance performance, while Token-DiFR maintains high detection accuracy across batch sizes.

2023) prompts. These models cover a wide range of differences between models such as hidden dimensions, training distributions and broader architectural differences such as being dense or mixture-of-experts. We collect approximately 1 million output tokens per configuration, with 9 configurations per model.

4.1. Inference Configurations

Reference configurations: The configuration advertised by the inference provider. Inference performed with this configuration represents the behavior expected by the user. We use vLLM as the inference engine with bf16 precision for model weights and activations, temperature of 1.0 for sampling, top-p of 0.95 nucleus sampling, top-k of 50 filtering, and a fixed PRNG seed.

Correct-but-noisy configurations. We simulate benign numerical noise by running inference with the reference configuration's parameters while varying GPU type, parallelism strategy, and inference engine: (i) H200 GPUs with 4-way tensor parallelism (TP-4), (ii) A100 GPUs with no tensor parallelism, (iii) A100 GPUs with TP-4, and (iv) H200 GPUs using HuggingFace instead of vLLM. These variations introduce floating-point noise due to differences in GPU architecture, kernels, and communication patterns. Together with the reference configuration, these form the null distribution for calibrating our statistical tests.

Pooling scores from multiple honest implementations is scientifically useful but introduces deployment overhead. In practice, a simpler approach is to use controlled perturbations such as FP8 KV cache quantization on a single reference implementation to define a tolerated noise band.

Incorrect configurations. To evaluate whether our metrics can identify actual misconfigurations in the inference process, we manually introduce four separate misconfigurations, each of which modifies exactly one parameter of the reference configuration, leaving the rest unaltered: (i) FP8 KV cache quantization. (ii) 4 bit model quantization instead of bf16. (iii) Temperature of 1.1 instead of 1.0. (iv) Incorrect random seed usage. We additionally explore a simulated sampling bug that samples an incorrect token 1% of the time, with further details in Appendix B. All incorrect configurations run on H200 GPUs with no tensor parallelism. Additional details are provided in Appendix F.

4.2. Verification Methodology

Token dataset generation. We first generate output sequences using each of our reference and correct-but-noisy configurations (our "honest" configurations) and each incorrect configuration. We then implement verification by running a single forward pass over each sequence using the reference configuration, measuring the divergence-from-reference at each token position using Token-DiFR and Activation-DiFR (as described in Section 3) as well as some baseline methods such as cross-entropy, token log-rank, and TOPLOC. We partition our dataset into training (50%) and test (50%) sets at the token level.

Feature engineering. We first multiply features by -1 if need be to ensure that higher scores indicate greater divergence-from-reference. We then winsorize (clip values to a chosen percentile) the continuous scores computed by Token-DiFR, Activation-DiFR, and TOPLOC at the 99th, 99.9th, 99.99th, and 99.999th percentiles found in the training split of the honest set, evaluating each winsorized variant separately to investigate the effect this preprocessing step has on batch-level classification performance. For Token-DiFR features, values are set to infinity when the inference provider's claimed token is filtered out by top-p or top-k at verification time. We exclude these infinite values when

	Llama 3.1 8B (H200)		Qwen3 30B A3B (A100)		Qwen3 30B A3B (H200)	
AUC $\geq \tau$	Act-DiFR	TOPLOC	Act-DiFR	TOPLOC	Act-DiFR	TOPLOC
0.95	**0.125**	0.219	**1.438**	7.25	**0.156**	0.312
0.99	**0.219**	0.406	**2**	14.5	**0.219**	0.438
0.999	**0.312**	0.625	**7.25**	32	**0.281**	0.562
0.9999	**0.375**	0.75	**30**	54	**0.375**	0.75

Table 1. **Communication cost** (bytes per token, lower is better) required to detect FP8 KV-cache quantization at different AUC levels (<1% FPR). For each setting, we sweep projection dimension k and logging stride J, reporting the minimum bytes-per-token needed to reach AUC $\geq \tau$ (see Appendix J). Bold entries indicate Activation-DiFR Pareto-dominates TOPLOC at each threshold.

computing percentiles, and then clip the infinite values to the corresponding percentile threshold.

Detector calibration. We randomly sample batches of tokens without replacement from each configuration and compute the mean score per batch, sweeping from 1 to 10,000 tokens. We then train binary classifiers on aggregated scores from the training set to discriminate between honest and incorrect configurations. This search is computationally inexpensive, requiring only one-dimensional classifiers on pre-computed scores. The optimal clipping value varies across models and deployment environments; we find that clipping at the 99.9th percentile generally works well for quantization detection, while more aggressive 99.999th-percentile clipping is better for rare bug detection. We find that choosing an appropriate clipping threshold can improve sample efficiency by more than 5×.

Performance evaluation. We report area under the curve (AUC) at 1% false positive rate (FPR), reflecting deployment scenarios where false positives may incur significant costs. If any features were found to have lower divergence-from-reference in the incorrect set than in the calibration set, then we do not train any downstream classifiers using that feature, setting their AUC to 0.5 (random chance). This mirrors realistic deployments: when the null distribution already spans many GPUs and implementations, a provider whose setup happens to match the verifier's (and therefore looks *more* consistent than average) should not be penalized, so we only treat configurations as suspicious when their scores are more divergent than the pooled baseline.

5. Results

We evaluate Token-DiFR and Activation-DiFR across three models (Llama 3.1 8B, Qwen3-8B, and Qwen3-30B-A3B) and nine configurations per model, spanning correct-but-noisy environments and a range of misconfigurations. Our main findings are:

1. When the verifier closely matches the provider's hardware and inference stack, Token-DiFR is a strong default that reliably detects both large and small misconfigurations across all models. Unlike cross-entropy, it is robust to simple adversarial manipulations.
2. When we deliberately pool together diverse "honest" implementations of the same model (different GPUs, parallelism setups, and inference engines), Token-DiFR continues to detect large problems (such as 4 bit quantization, FP8 KV cache quantization, or incorrect seeds) but sometimes struggles with the smallest deviations (temperature off by 0.1).
3. Activation-DiFR achieves near-saturated detection performance for quantized models using very small batch sizes (1-4 tokens), and Pareto-dominates TOPLOC in communication cost versus detection accuracy.

5.1. Token-DiFR in Matched vs. Mismatched Environments

For Llama 3.1 8B and Qwen3-8B, Token-DiFR maintains strong performance across all settings. As shown in Figure 2, Token-DiFR remains robust to adversarial temperature tuning that defeats cross-entropy-based detectors. We obtain near-perfect AUC for all misconfigurations by aggregating over a few thousand tokens: large deviations such as 4-bit quantization are detected after 100–500 tokens, while subtler changes such as FP8 KV-cache quantization require 1,000–5,000 tokens.

Seed synchronization tightly constrains token generation. Distributional verification methods permit countless valid generations matching the expected distribution, creating vulnerability to simple adversarial strategies such as temperature tuning (Figure 2). By contrast, we find that across all models and honest configurations, over 98% of generated tokens exactly match the verifier (Table 6). This leaves providers with almost no room for deviation.

Token-DiFR requires many tokens for minor misconfigurations because they often produce identical outputs: empirically, over 95% of generated tokens exactly match the trusted configuration for FP8 KV Cache quantization (see Section L). Detection therefore relies on aggregating evidence across many tokens rather than any single generation.

For Qwen3-30B-A3B, pooled honest environments make the smallest misconfigurations hard to detect with a sin-

gle margin statistic. We found that not all "honest" implementations behaved equally: the HuggingFace implementation yielded much larger benign Token-DiFR gaps than vLLM-based configurations, with high-percentile values comparable to our FP8 KV cache quantization setting (see Table 4). This raises a conceptual point: different inference implementations can introduce real behavioral differences, and it is up to the end user whether to treat those as acceptable noise. We exclude HuggingFace from the default pooled results and present them separately in Appendix N.

In the pooled vLLM-only setting, Token-DiFR still detects large deviations (4-bit quantization, incorrect seeds, FP8 KV-cache quantization) with near-perfect AUC within a few thousand tokens. However, the smallest deviations—a temperature shift from 1.0 to 1.1 and the simulated k=2 bug (results in Figure 3) remain difficult to separate from the broadened null distribution, while cross-entropy succeeds on these cases. In Appendix H we show that alternative aggregation schemes emphasizing rare large deviations restore strong Token-DiFR performance.

Matched verifier and provider environments restore uniformly strong performance for Token-DiFR. When we restrict attention to matched environments, Token-DiFR is consistently strong for all three models. Even the smallest misconfigurations for Qwen3-30B-A3B are reliably detected with a few thousand tokens, as shown in the appendix (see Appendix Figure 13).

5.2. Activation-DiFR and Communication Efficiency

Activation-DiFR provides a continuous verification signal even when tokens match, enabling detection at much smaller batch sizes than Token-DiFR. For Llama 3.1 8B and Qwen3-8B, Activation-DiFR reaches near-saturated detection of 4-bit quantization with just 1–4 tokens when the projection dimension k is sufficiently large (Appendix Figure 4). For Qwen3-30B-A3B, 4-bit quantization is detected within approximately 4 tokens, but detecting FP8 KV-cache quantization requires 16–20 tokens due to the smaller activation-level perturbation.

We compare activation-based methods in terms of communication cost (bytes per token) required to reach a given detection threshold, sweeping the projection dimension k and logging stride J (see Appendix J for full Pareto analysis). Table 1 shows that Activation-DiFR consistently Pareto-dominates TOPLOC, achieving roughly a $2\times$ reduction across all settings. On matched H200 hardware, it reaches AUC ≥ 0.9999 with under 0.5 bytes per token.

5.3. Case Study: Auditing Production Llama 3.1 8B Providers

To demonstrate practical applicability, we conducted a small audit of five third-party providers serving Llama 3.1 8B via public APIs. We queried each provider with 2,000 prompts at temperature zero, generating up to 500 tokens per prompt (500k tokens total per provider), and computed Token-DiFR scores against a local bf16 reference. At temperature zero, no seed synchronization is required, as the verifier simply checks argmax agreement.

Figure 5 shows that provider scores vary substantially. Groq's deployment achieves scores between our bf16 and fp8-KV reference bands despite running on custom LPU hardware rather than NVIDIA GPUs, providing evidence that Token-DiFR is robust to hardware differences beyond our controlled experiments. Silicon Flow's scores align with the fp8 reference band, consistent with their advertised fp8 configuration. Cerebras and DeepInfra exhibit larger divergence. Manual inspection revealed both use an older Llama chat-template format, which demonstrates how Token-DiFR can surface specification mismatches that affect downstream behavior. This case study required no provider cooperation (only public API access) demonstrating that practitioners can begin spot-checking open-weight deployments today.

6. Discussion

Practical Deployment. As described in Section 3.1, Token-DiFR supports immediate deployment via temperature-zero spot-checking and retroactive auditing via seed synchronization. We discuss standardization recommendations and additional deployment considerations, including API-only verification without local weights, in Appendix A.

Comparison with Distributional Methods. Distribution-based methods such as MMD (Gao et al., 2025) and RUT (Zhu et al., 2025) test whether outputs are statistically consistent with a reference distribution. However, they require substantial sampling (RUT uses 100 reference generations per prompt), are vulnerable to adversarial manipulation via temperature tuning (Figure 2), and struggle with subtle misconfigurations. Zhu et al. (2025) report that RUT fails to reliably distinguish 8-bit quantized models. In contrast, by conditioning on a shared seed, Token-DiFR permits far fewer degrees of freedom.

Limitations. Token-DiFR and Activation-DiFR require access to model weights, enabling verification of open-source models but not closed-weight providers. Practical deployment also requires knowledge of the provider's sampling procedure; for open-source frameworks like vLLM this is straightforward, but custom implementations would require per-provider verification functions.

References

Anthropic. A postmortem of three recent issues. https://www.anthropic.com/engineering/a-postmortem-of-three-recent-issues, 2025. Accessed: 2025-10-04.

Chen, B., Zheng, M., Liu, T., Piskac, R., and Shao, Z. zktorch: Privacy-preserving deep learning with zero-knowledge proofs, 2025. URL https://arxiv.org/abs/2502.00226.

Chen, C., Borgeaud, S., Irving, G., Lespiau, J.-B., Sifre, L., and Jumper, J. Accelerating large language model decoding with speculative sampling, 2023. URL https://arxiv.org/abs/2302.01318.

Gao, I., Liang, P., and Guestrin, C. Model equality testing: Which model is this api serving?, 2025. URL https://arxiv.org/abs/2410.20247.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., et al. The llama 3 herd of models. 2024. URL https://arxiv.org/abs/2407.21783.

Gumbel, E. J. *Statistical Theory of Extreme Values and Some Practical Applications: A Series of Lectures*, volume 33 of *Applied Mathematics Series*. U.S. Department of Commerce, National Bureau of Standards, Washington, D.C., 1954.

H4, H. Ultrachat 200k. https://huggingface.co/datasets/HuggingFaceH4/ultrachat_200k, 2023. Filtered subset of 1.4M dialogue dataset.

He, H. and Lab, T. M. Defeating nondeterminism in LLM inference. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20250910. https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/.

Huijben, I., Kool, W., Mooij, J. M., de Vries, B., and van Sloun, W. J. J. A review of the gumbel-max trick and its extensions for discrete stochasticity in machine learning. *arXiv preprint arXiv:2110.01515*, 2021.

Johnson, W. B. and Lindenstrauss, J. Extensions of Lipschitz mappings into a Hilbert space. In *Contemporary Mathematics*, volume 26, pp. 189–206. American Mathematical Society, 1984.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

Leviathan, Y., Kalman, M., and Matias, Y. Fast inference from transformers via speculative decoding, 2023. URL https://arxiv.org/abs/2211.17192.

Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle-3: Scaling up inference acceleration of large language models via training-time test, 2025. URL https://arxiv.org/abs/2503.01840.

Moonshot AI. K2 vendor verifier. https://github.com/MoonshotAI/K2-Vendor-Verfier, 2025. Accessed: 2025-10-04.

Ong, J. M., Ferrante, M. D., Pazdera, A., Garner, R., Jaghouar, S., Basra, M., Ryabinin, M., and Hagemann, J. Toploc: A locality sensitive hashing scheme for trustless verifiable inference, 2025. URL https://arxiv.org/abs/2501.16007.

Rinberg, R., Karvonen, A., Hoover, A., Reuter, D., and Warr, K. Verifying llm inference to prevent model weight exfiltration, 2025. URL https://arxiv.org/abs/2511.02620.

Singh, A., Virga, F., Smith, S., and Hogan, S. LOGIC: Trustless Inference through Log-Probability Verification. https://inference.net/blog/logic, 2025. Context Labs Blog, November 5, 2025.

Sun, H., Li, J., and Zhang, H. zkllm: Zero knowledge proofs for large language models, 2024. URL https://arxiv.org/abs/2404.16109.

Vieira, T. The gumbel-max trick. https://timvieira.github.io/blog/post/2014/07/31/gumbel-max-trick/, 2014. Accessed: 2025-10-06.

vLLM Contributors. vLLM gumbel-max sampler implementation. https://github.com/vllm-project/vllm/blob/10d765482d19abfab6c66b5f815720a66aa9de42/vllm/v1/sample/ops/topk_topp_sampler.py#L252, 2024. Line 252.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. Huggingface's transformers: State-of-the-art natural language processing, 2020. URL https://arxiv.org/abs/1910.03771.

Yang, A. et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Zhang, L., Wang, X., Huang, Y., and Xu, R. Learning harmonized representations for speculative sampling, 2025. URL https://arxiv.org/abs/2408.15766.

Zhu, X., Ye, Y., Qiu, T., Zhu, H., Tan, S., Mannan, A., Michala, J., Popa, R. A., and Neiswanger, W. Auditing black-box llm apis with a rank-based uniformity test, 2025. URL https://arxiv.org/abs/2506.06975.

A. Practical Deployment Considerations

A.1. Practical Recommendations

Token-DiFR is a strong default metric, while cross-entropy is a practical fallback when RNG seed synchronization is not available. When deterministic sampling is unavailable—for instance, when providers cannot or will not expose seed synchronization—cross-entropy provides a reasonable alternative. While it requires approximately 5-10$\times$ more samples to achieve equivalent detection performance for certain misconfiguration types such as quantization, it remains effective for detecting major sampling bugs, incorrect temperature configurations, and quantized models.

In practice, we expect practitioners to maintain a small family of simple one-dimensional detectors rather than rely on a single score. Once logits are available, Token-DiFR margins, their clipped variants, likelihood-style transformations, and cross-entropy are all cheap to compute and easy to threshold. In our experiments, more aggressive clipping (at the 99.999th percentile) tends to work better for detecting rare sampling bugs, while milder clipping (at the 99.9th percentile) improves sensitivity to subtle but systematic changes such as quantization. Cross-entropy also remains a useful auxiliary signal, particularly for bug detection in settings like Qwen3-30B-A3B where small temperature shifts have strong effects on average log-likelihood. In realistic deployments, it is natural to train and monitor several such detectors in parallel, each tuned to a different class of potential failure modes.

A simple practical method for calibrating benign noise. In our experiments we build a pooled honest baseline by combining configurations across multiple GPUs, parallelism strategies, and inference engines. This setting is useful for scientific understanding, but it introduces complexity which is not required for deployment. A simpler pattern is to fix a single reference implementation and then use controlled perturbations such as FP8 KV-cache quantization or a small temperature shift (for example, $T = 1.1$ instead of $T = 1.0$) to estimate the level of numerical variation one is willing to regard as benign. Configurations that induce larger shifts are flagged as misconfigured, without requiring the verifier to enumerate or explicitly calibrate every possible implementation the provider might be using.

Cross-entropy (the baseline) has several conceptual limitations that practitioners should consider. Most fundamentally, it discards the signal available in the sampling seed. When conditioning on a known seed, the token generation process becomes nearly deterministic, with very few potential tokens at any step (this near-determinism also means that seed-conditioned sampling verification may be useful for detecting steganographic channels). In contrast, there are many potential generations that could achieve an average cross-entropy value.

As demonstrated in Figure 2, a cross-entropy based classifier can be defeated by a simple attack where the adversary tunes the temperature to obtain the correct mean cross-entropy. Additionally, cross-entropy is significantly more sensitive to both the sampling configuration (temperature, top-p) and the entropy of the output distribution itself. For example, generations of sequential numbers will produce systematically lower cross-entropy values than generations of random numbers, even when both are correctly sampled, because sequential outputs have inherently lower entropy. This distribution-dependent behavior can complicate classification.

Activation-DiFR provides additional assurance for critical applications and does not require seed synchronization. Token-DiFR alone is sufficient for most verification scenarios, reliably detecting misconfigurations with batch sizes of 5,000-10,000 tokens without requiring any modifications to standard inference implementations or communication overhead. However, at these batch sizes, Token-DiFR verifies the integrity of the batch as a whole rather than individual sequences within the batch—to verify each sequence independently would require smaller batch sizes or more tokens per sequence.

Activation-DiFR provides much higher sensitivity to subtle deviations such as minor quantization errors, detecting issues with as few as 1 token. For applications requiring the highest confidence, we recommend combining Activation-DiFR with Token-DiFR to jointly verify both the sampling procedure and the integrity of the forward pass. In addition, unlike Token-DiFR, Activation-DiFR does not require RNG seed synchronization, making it applicable even when providers cannot expose sampling algorithms or seeds.

A.2. Deployment Considerations

Token-DiFR and Activation-DiFR require access to model weights. These methods enable model providers to verify their own inference traffic and allow users to verify open-source models, but cannot verify inference from providers serving closed-weights models.

API-only verification is possible. For large models where local inference is impractical, verification can proceed without access to model weights. Some inference providers (e.g., Tinker and Fireworks) return log-probabilities for input tokens via their APIs. To use this for verification, a verifier first generates a set of reference outputs from a trusted source: either a local deployment or an authoritative provider such as the model developer's official API (e.g., Moonshot AI's deployment of Kimi). These reference outputs are stored once. To audit a third-party provider, the verifier submits the stored outputs as input tokens and

compares the provider's returned log-probabilities against those from the trusted reference. This approach is particularly valuable for frontier-scale models such as Kimi K2 (1 trillion parameters), where local hosting is inconvenient.

Token-DiFR can be deployed today for spot-checking using temperature zero. At temperature zero (greedy decoding), Token-DiFR requires no sampling synchronization, as the verifier simply recomputes the forward pass and checks whether the provider selected the argmax token at each position. This makes immediate deployment feasible for users who wish to spot-check providers by requesting greedy generations, as we illustrate in a small in-the-wild case study on third-party Llama 3.1 8B providers (Appendix G). However, this would not work for typical production usage at non-zero temperatures.

In practice, temperature-zero spot checking would successfully detect honest misconfigurations (e.g., a provider accidentally using an incorrect tokenization format). However, it is vulnerable to selective cheating, as a malicious provider could serve high-quality inference only at temperature zero while using degraded models for non-zero temperature requests. Therefore, verifying production usage at non-zero temperatures requires access to the provider's sampling procedure and the ability to synchronize random seeds.

Practical deployment requires knowledge of the provider's sampling procedure. For open-source inference frameworks like vLLM, this requirement is straightforward—the sampling code is publicly available and can be directly replicated by verifiers. The core sampling logic typically consists of only a few dozen lines of code, making verification implementation tractable. Providers must also expose an API parameter allowing clients to specify per-request sampling seeds with a known noise generation procedure. For example, vLLM uses a per-request `torch.Generator` passed to `exponential_()` for Gumbel sampling.

We recommend that open-source inference providers standardize on a common sampling algorithm. Open source providers can enable verification either by sharing their sampling procedure with verifiers or by adopting a standardized implementation (such as vLLM's sampling implementation). If providers use custom sampling procedures, verification requires implementing a custom verification function per provider. Given that sampling logic is typically not a performance bottleneck, standardization imposes minimal overhead while enabling straightforward verification across providers. Alternatively, providers could offer a standard sampling mode alongside their custom implementations to support verification use cases without constraining their default behavior.

Practical Deployment via Random Sampling While our framework requires the verifier to regenerate outputs, this does not impose prohibitive costs in practice. Not every user needs to verify every generation, as random sampling allows sporadic spot-checks to detect systemic issues. Since systematic misconfigurations affect outputs consistently, relatively small sample sizes suffice to provide high confidence about overall inference quality. Modern LLMs generate tokens at costs of just a few dollars per million tokens, making verification overhead negligible relative to total inference volume. The amount of verification needed is a tradeoff of cost versus confidence.

B. Bug Detection Simulation

Sampling Bug To evaluate Token-DiFR's ability to detect implementation bugs, we simulate sampling bugs by introducing controlled deviations from the intended sampling process. In the reference configuration, tokens are sampled according to the model's softmax distribution at temperature T. In the buggy configurations, 1% of the time, the sampler instead ignores the model's logits and uniformly selects one of the top-k tokens (for $k \in 2, 32$). For the remaining 99% of tokens, normal sampling proceeds unchanged.

As seen in Figure 3, Token-DiFR can typically detect these simulated bugs, and cross-entropy is a strong baseline that sometimes outperforms Token-DiFR in the bug detection setting.

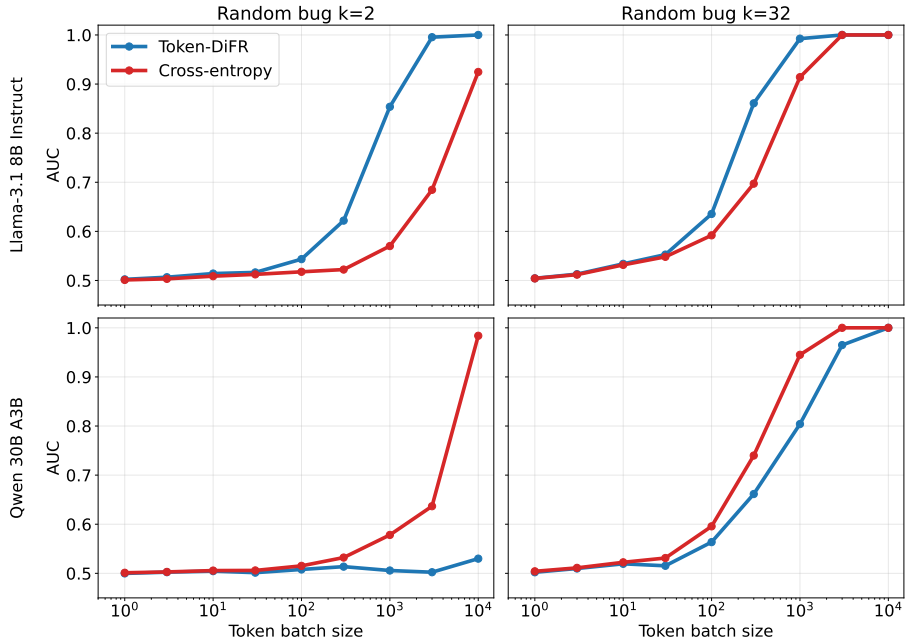


Figure 3. Detection of simulated sampling bugs for Llama 3.1 8B and Qwen3-30B-A3B. We introduce a bug that, with probability 1% per token, ignores the model logits and instead samples uniformly from the top-k tokens ($k \in \{2, 32\}$), and otherwise samples correctly. The curves show AUC at 1% FPR as a function of batch size for cross-entropy and Token-DiFR variants. For Llama 3.1 8B, Token-DiFR detects both bug settings with modest batch sizes. For Qwen3-30B-A3B, the simple mean-pooled margin score underperforms cross-entropy in the $k = 32$ case and fails to separate $k = 2$ bugs from the pooled honest baseline. We pool per-token scores by taking their mean for consistency with our other figures, but in Appendix H we show that simple alternative aggregation schemes which emphasize rare large deviations restore strong Token-DiFR performance in this regime, and we recommend that practitioners consider monitoring multiple aggregation strategies in parallel.

C. Activation-DiFR: Extended Results

Here we present extended batch-size scaling results. Activation-DiFR once again Pareto-dominates TOPLOC and typically saturates detection performance using batch sizes as small as 4 tokens.

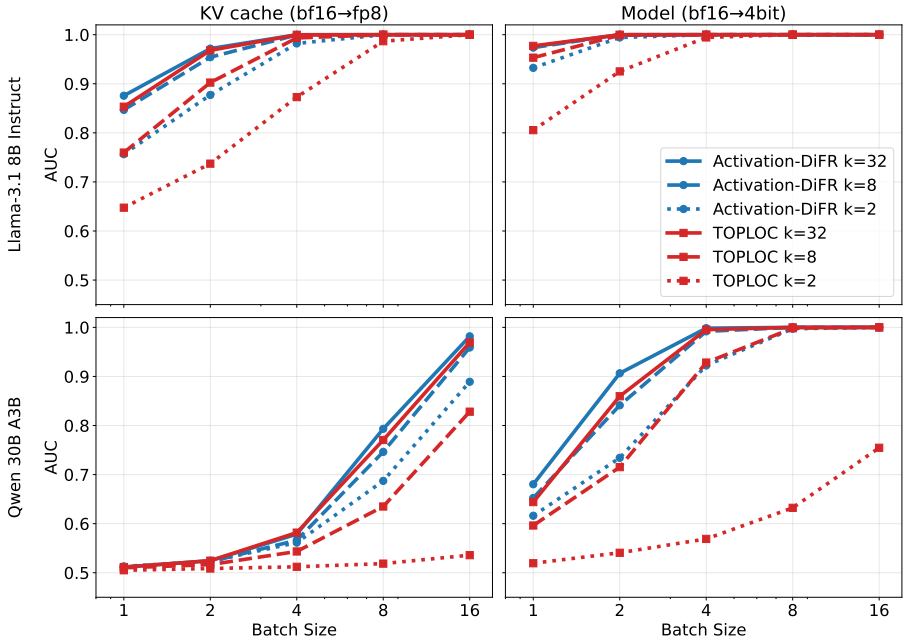


Figure 4. **Activation-DiFR detects quantization with small batches and outperforms TOPLOC at fixed communication cost.** We plot AUC at $< 1\%$ FPR as a function of batch size when distinguishing the reference configuration from 4-bit model quantization and FP8 KV-cache quantization, using activation fingerprints with projection dimension $k \in \{2, 8, 32\}$. Across both Llama 3.1 8B and Qwen3-30B-A3B, Activation-DiFR reaches near-saturated detection with small batches (for example, FP8 KV-cache quantization on Llama 3.1 8B is detected with roughly 4 tokens at $k = 8$) and, at any fixed k, matches or exceeds TOPLOC's detection accuracy. In the mismatched Qwen3-30B-A3B setting, classification still eventually saturates but requires larger batch sizes, similar to the increased difficulty observed for Token-DiFR under pooled honest environments.

D. Token-DiFR Score Transformations

In the main text, we use the margin-based Token-DiFR score as our default verification metric. For completeness, we also explored a small number of simple score transformations on top of the raw logit gaps. Empirically, these variants yield at most modest improvements, mostly in the bug detection setting, and generally do not justify the added complexity or hyperparameters, so we do not treat them as primary metrics in the body of the paper. We nevertheless describe them here and include them in the remaining appendix plots.

Likelihood-based variant. The clipped logit difference can also be converted into a likelihood-style score by introducing a simple noise model for the logits. We treat the logit gap Δ between the claimed token and the verifier's best token as being modified by Gaussian noise with standard deviation σ, and map it to a standardized score $z = -\Delta/\sigma$. We then define a likelihood score

$$\mathcal{L}(t^*) = \log \Phi(z),$$

where Φ is the standard normal cumulative distribution function. The parameter σ can be estimated from a small calibration set or swept over a short grid as a single additional hyperparameter for improved classification performance.

Exact-match variant. We also consider a binary exact-match metric that assigns a score of 1 if the provider and verifier sample the same token under a shared seed and 0 otherwise. This variant is hyperparameter-free, but it produces an inherently worse classifier as it discards significant amounts of information contained in the logit differences.

E. Additional Verification Algorithms

E.1. Monte Carlo Simulation

For sampling algorithms where analytical likelihood computation is difficult or intractable—which could potentially include certain speculative decoding schemes—Monte Carlo simulation provides a fully general verification approach. The idea is to explicitly model the effect of numerical noise on the logits through repeated simulation of the sampling process.

Given the verifier's computed logits $\ell \in \mathbb{R}^V$ and a claimed token t^*, we sample Gaussian noise $\xi \sim \mathcal{N}(0, \sigma^2 I)$ and run the provider's sampling algorithm on perturbed logits $\ell + \xi$ for N trials (e.g., $N = 1000$). The verification score is simply the empirical frequency:

$$\mathcal{L}_{\text{MC}}(t^*) = \log\left(\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}[\text{sample}(\ell + \xi_i) = t^*] + \varepsilon\right)$$

where $\varepsilon > 0$ ensures numerical stability. To reduce computational cost, we apply noise only to the top-k logits (we use $k = 100$ in practice) and set remaining logits to $-\infty$.

While Monte Carlo simulation is somewhat more computationally expensive than analytical methods and has limited resolution for distinguishing between low-probability events, it offers a simple and general approach that works with any sampling algorithm and is still relatively cheap in practice. Empirically, we find it often achieves performance close to our analytical likelihood scores.

Algorithm 1 Inverse-Probability-Transform (IPT) Sampler

Require: Seed s, probability vector $\mathbf{p} \in [0,1]^V$ with $\sum_{t=1}^{V} p_t = 1$

$\mathbf{C} \leftarrow \textsc{CumulativeSum}(\mathbf{p})$ {Convert PDF $\mathbf{p}$ to CDF $\mathbf{C}$}

$u \leftarrow \textsc{Uniform}(s)$ $\min\{t \in \{1,\ldots,V\} : C_t > u\}$ {binary search: find smallest t s.t. $C_t > u$}

E.2. Inverse Probability Transform Sampling

Algorithm 1 presents the Inverse Probability Transform (IPT) sampling procedure, a common alternative to Gumbel-Max sampling for token generation, used in some inference stacks like HuggingFace (Wolf et al., 2020). Given a probability vector $\mathbf{p} \in [0,1]^V$ (obtained by applying temperature scaling, top-k and top-p filters, and softmax to the logit vector), IPT sampling draws a uniform random variable $u \sim \text{Uniform}(0,1)$ and returns the smallest token index t such that the cumulative distribution function satisfies $C_t > u$.

IPT-likelihood score for IPT Verification We propose Token-IPT-DiFR, a method for estimating the IPT-likelihood score. For a claimed token index $g \in \{1,\ldots,V\}$ produced by IPT with shared seed s, the claimed token's interval on the CDF is

$$I_g = (C_{g-1}, C_g] \quad \text{with width} \quad w_g = C_g - C_{g-1}.$$

To account for numerical noise in the computed probabilities, we model u as corrupted by Gaussian noise with standard deviation σ. We place a Gaussian kernel on the uniform draw and evaluate the probability mass that falls inside I_g. Let

$$Z \sim \mathcal{N}(u, \sigma^2),$$

where u is the shared uniform draw and $\sigma > 0$ is a calibration parameter that captures effective noise in the verifier–provider pipeline. The *Token-IPT-DiFR* computes

$$\mathcal{L}_{\text{CG}}(g \mid u, \mathbf{C}, \sigma) = \log\Big(\Phi\big(\tfrac{C_g - u}{\sigma}\big) - \Phi\big(\tfrac{C_{g-1} - u}{\sigma}\big) + \varepsilon\Big),$$

where Φ is the standard normal CDF and $\varepsilon > 0$ is a small constant for numerical stability.

Because temperature is applied to scale the logits, the σ or scores may have to be scaled as well to enable use of the same classifier at multiple temperature values.

F. Implementation Details for Incorrect Configurations

For reproducibility, we provide the exact vLLM arguments used to implement each misconfiguration. Temperature, seed, and top-p are standard `SamplingParams` arguments in vLLM. The quantization configurations are specified during model initialization with the following arguments:

- **4-bit quantization:** `{"quantization": "bitsandbytes"}`
- **FP8 model quantization:** `{"quantization": "fp8"}`
- **FP8 KV cache quantization:** `{"kv_cache_dtype": "fp8", "calculate_kv_scales": True}`

For Qwen3-30B-A3B, we used the pre-quantized model `RedHatAI/Qwen3-30B-A3B-quantized.w4a16` for our 4-bit quantization as bitsandbytes quantization was not supported for this model.

F.1. Non-Zero Baseline Detection Rates

Our experimental design intentionally mimics real-world deployment: the trusted distribution combines outputs from multiple GPU types and configurations (H200 with TP-4, A100 with and without TP-4, and H200 with HuggingFace), while each untrusted provider uses a single GPU setup. This reflects realistic scenarios where verifiers may aggregate trusted outputs from multiple sources, but each individual provider operates with a fixed hardware configuration.

This setup introduces a detection artifact: metrics like cross-entropy show small but consistent differences across GPU types and parallelism strategies. As a result, classifiers can achieve slightly elevated detection rates even for configurations that should be indistinguishable from the reference (e.g., classifying between different random seeds with the cross-entropy metric). The classifier exploits hardware-induced biases rather than actual configuration differences.

As seen in Figure 1, these baseline effects are minor and do not prevent detection of substantive misconfigurations, which show much higher detection rates.

G. In-the-wild audit of public Llama 3.1 8B providers

Existing work on RUT (Zhu et al., 2025) and MMD (Gao et al., 2025) evaluate existing open-source inference services; in similar a vein, to evaluate the usefulness of Token-DiFR, we ran a small case study on several third-party providers that serve Llama 3.1 8B behind a public API.[1]

Setup. We first calibrated four local reference configurations of `meta-llama/Llama-3.1-8B-Instruct` under vLLM: a bf16 baseline, an fp8-quantized baseline, an fp8 KV-cache quantized baseline, and a 4-bit-quantized baseline. For each configuration, we sampled 2,000 prompts from our evaluation distribution, generated up to 500 tokens per prompt at temperature zero (greedy decoding), and computed per-token Token-DiFR scores by replaying the same generations under the bf16 reference and measuring the clipped post-Gumbel logit margin described in Section 3.3. This yields a mean Token-DiFR score for each configuration, which we treat as a reference band for bf16-like, fp8-like, fp8-KV-like, and 4-bit-like behavior.

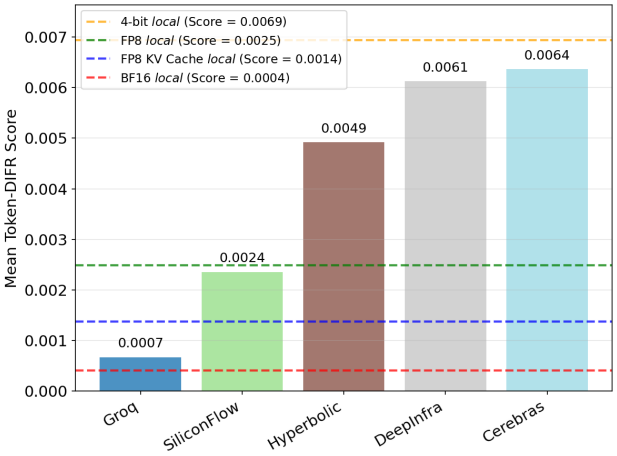
We then queried five public Llama 3.1 8B deployments via the API at temperature zero, using the same 2,000 prompts and generation length. For each provider, we treated its greedy outputs as the claimed tokens and computed Token-DiFR scores against our local bf16 reference in exactly the same way as for the controlled baselines, aggregating over approximately 5×10^5 output tokens per provider.

Findings. Scores vary substantially across providers. Figure 5 shows that Groq's Token-DiFR and cross-entropy scores fall between our bf16 and fp8-KV reference bands, which is consistent with their advertised near-bf16 configuration and gives us high confidence that they are serving something close to the intended specification. Silicon Flow lies near the fp8 reference band, which is consistent with their advertised fp8 configuration. The remaining providers (Hyperbolic, Cerebras, and DeepInfra) exhibit higher scores and therefore diverge more from our bf16 reference.

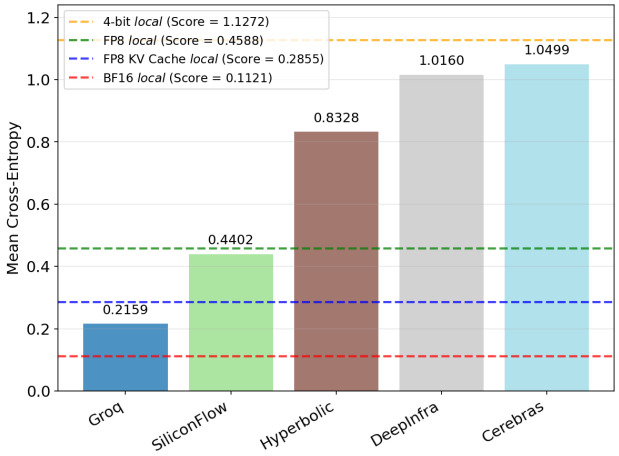
Low Token-DiFR and cross-entropy scores indicate tight adherence to the chosen reference specification, while higher scores indicate systematic differences that may arise from either genuinely degraded inference or benign specification choices such as alternative tokenizers or chat templates. For Cerebras and DeepInfra, manual inspection indicates that both use an earlier Llama chat-template format with an older system prompt, which likely accounts for a substantial fraction of the observed divergence, although we do not attempt to separate this from other implementation or configuration differences. We emphasize that higher scores in this case study should be interpreted as evidence of a different effective specification rather than as a direct claim about overall model quality.

In the temperature-zero setting, both Token-DiFR and cross-entropy are simple to compute without seed synchronization and behave similarly in practice, making them practical tools for spot-checking open-weight deployments and gaining confidence that advertised configurations are being followed. Unlike the non-zero-temperature setting considered in our adversarial experiments, it is harder to cheat cross-entropy with simple tricks such as temperature tuning, since greedy decoding largely fixes the output distribution. In principle, a malicious provider could still search for low cross-entropy generations (for example, by running beam search or reranking multiple candidates), but this would incur significant additional inference cost and latency, making such attacks less attractive in typical deployment scenarios.

[1] Accessed in November 2025; implementations and configurations may have changed since then.



(a) Token-DiFR Scores



(b) Cross-Entropy Scores

Figure 5. **Token-DiFR and cross-entropy can audit in-the-wild Llama 3.1 8B deployments at temperature zero.** Low scores indicate tight adherence to the chosen reference specification, while higher scores indicate divergence, which can stem from either genuinely degraded inference (for example, heavy quantization) or benign specification differences such as alternative tokenizers or chat-template formats. In this non-adversarial, temperature-zero setting, Token-DiFR and cross-entropy behave similarly and are equally simple to compute without seed synchronization.

H. Alternative pooling strategies for rare bug detection

In the main text, our default Token-DiFR detector aggregates per-token scores by taking the mean of clipped margins across all tokens in a batch. This works well for systematic misconfigurations such as quantization or temperature shifts, but rare bug regimes, where only a small fraction of tokens are affected, are particularly sensitive to how per-token scores are pooled.

To illustrate this, we consider a simple tail-focused pooling scheme applied to the simulated bug experiments in Figure 3. Given a set of per-token Token-DiFR margin scores $\{s_j\}$ for a batch, we first clip scores at the 99.999th percentile and then set all scores below the 99.99th percentile to zero. The final batch statistic is the mean of these transformed scores. This clipping suppresses benign fluctuations while concentrating weight on rare large deviations.

On Qwen3-30B-A3B with the $k = 2$ sampling bug, this simple transformation of the same per-token Token-DiFR scores is sufficient to recover strong bug detection performance: at FPR $= 1\%$, the detector reaches AUC $= 1.0$ with $10{,}000$ tokens.

Practically, once a verifier has access to per-token scores, it is cheap to compute several pooled statistics in parallel (for example, mean, clipped mean, tail-focused pooling, or likelihood-style transforms) and either threshold them separately or combine them with a lightweight classifier. We therefore view Token-DiFR as a family of detectors parameterized both by the per-token score and the pooling rule, and recommend that practitioners select or monitor multiple aggregation strategies based on the failure modes they care about.

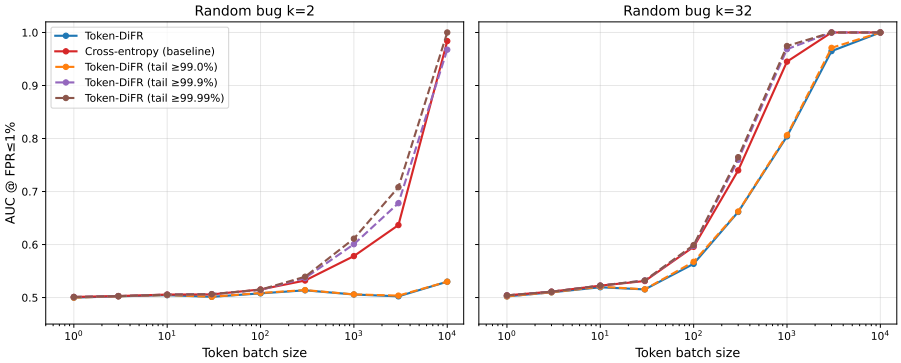


Figure 6. **Effect of tail-focused clipping on rare bug detection.** For the Qwen3-30B-A3B $k = 2$ sampling bug, we compare the original mean-pooled Token-DiFR margin to a tail-focused variant that clips scores at the 99.999th percentile and zeros scores below the 99.99th percentile. The tail-focused pooling restores strong performance, reaching AUC $= 1.0$ at 10,000 tokens.

I. Extension to Speculative Decoding

Speculative decoding (Leviathan et al., 2023; Chen et al., 2023; Zhang et al., 2025; Li et al., 2025) has emerged as a technique to accelerate LLM inference by using a smaller "draft" model to propose candidate tokens, which are then verified in parallel by the larger "target" model. This approach can achieve substantial speedups (often $2{-}3\times$) while preserving the exact output distribution of standard autoregressive sampling from the target model.

In the main body of this work, we focus our verification methodology on standard autoregressive sampling (Section 3.3), where a single model generates one token at a time. This choice reflects two practical considerations:

1. There are a variety of proposed speculative decoding algorithms, each of which may require a slightly modified verification algorithm.

2. At the time of writing, vLLM version 0.10 does not natively support speculative decoding.

Nevertheless, we recognize that speculative decoding represents an important class of real-world inference strategies. In this appendix, we demonstrate that our verification framework naturally extends to handle speculative decoding with minimal modifications. We sketch the necessary adaptations using the canonical algorithm from (Leviathan et al., 2023; Chen et al., 2023) as a concrete example.

I.1. Background: A Speculative Decoding Algorithm

While many subsequent methods have been proposed since the original proposal, in the speculative decoding work introduced by (Leviathan et al., 2023; Chen et al., 2023), a draft model generates K candidate tokens, which the target model then verifies in parallel. In the original work, for each candidate, the target model either accepts it, rejects and resamples from an adjusted distribution, or (if all candidates are accepted) samples one additional bonus token. This process maintains exact distributional equivalence to standard autoregressive sampling from the target model alone.

I.2. Verification Strategy

The key observation enabling verification is that *each sampled token is a function of the random seed and the model logits from previous tokens*—precisely the quantities available to the verifier. Unlike standard autoregressive sampling, however, this particular version of speculative decoding uses three distinct sampling modes:

1. **Accept**: Token accepted from draft proposal

2. **Reject**: Token resampled from adjusted target distribution $\max(q - p, 0)$

3. **Bonus**: Final token sampled when all draft tokens accepted

If the inference provider is using this given method of speculative decoding,then to enable verification, the inference server must record which mode was used for each output token. Given this metadata, the verifier:

1. Performs a single forward pass with both the draft and target models over the full input + output sequence to obtain p_i and q_i at all positions

2. For each token, computes the probability of observing that token under the recorded mode and seed (via analytical calculation or Monte Carlo simulation, using logit noise estimates for the draft and target models)

This approach preserves the efficiency advantages of our verification framework: the verifier performs only prefill operations (no sequential decode), and verification cost remains dominated by a single target model forward pass (draft models are typically small).

I.3. Generality and Limitations

The approach outlined here applies directly to a single speculative decoding algorithm. However, many variants exist (e.g. (Zhang et al., 2025; Li et al., 2025)), each with different acceptance criteria and resampling rules. Extending our verification framework to these methods would require identifying the sampling modes and recording which mode was used at each position.

However, our general technique of replaying inference with a reference implementation remains applicable regardless of the underlying sampling algorithm. The primary engineering challenge is ensuring that the inference server records sufficient metadata to make each token's sampling process reproducible by the verifier.

J. Activation Fingerprint Communication Cost Analysis

To analyze the trade-off between communication overhead and detection accuracy, we examine the operational scenario where a verifier must classify batches of 32 tokens as either correctly or incorrectly generated. We limit ourselves to the setting of detecting fp8 KV Cache quantization. For each token position within a batch, the verifier can choose to collect activation fingerprints at varying granularities, parameterized by k—the number of features extracted per token.

J.1. Experimental Setup

In this setting, the minimum communication cost occurs when using $k = 1$ on only a single token within each 32-token batch, yielding a cost of 1 byte per 32 tokens. At the other extreme, maximum verification fidelity is achieved by using $k = 64$ on all 32 tokens, incurring a cost of $64 \times 32 = 2{,}048$ bytes per 32 tokens. Between these endpoints, we evaluate all combinations of $k \in \{1, 2, 4, 8, 16, 32, 64\}$ and batch sizes $B \in \{1, 2, \ldots, 32\}$, computing the communication cost as $k \times B$ bytes per 32 tokens.

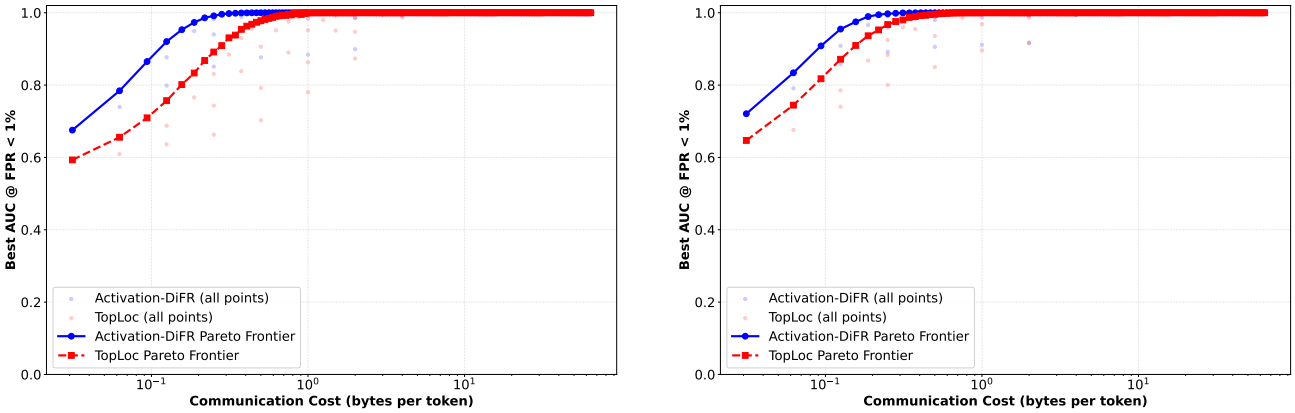
For each (k, B) configuration, we train a classifier on the training split using the methodology described in Section 4.2 and evaluate its true positive rate (TPR) at a fixed false positive rate of 1% on the held-out test set. This yields a collection of $(k \times B, \text{TPR})$ points representing different operating points along the accuracy-cost spectrum.

J.2. Pareto Frontier Analysis

We compute the Pareto frontier by identifying, for each communication cost level, the configuration achieving maximum TPR. Configurations not on this frontier are strictly dominated—there exists an alternative with equal or lower cost and strictly better detection performance. Figure 7 presents the Pareto frontiers for Activation-DiFR and TOPLOC across three model architectures, demonstrating that Activation-DiFR consistently dominates TOPLOC across all cost levels.

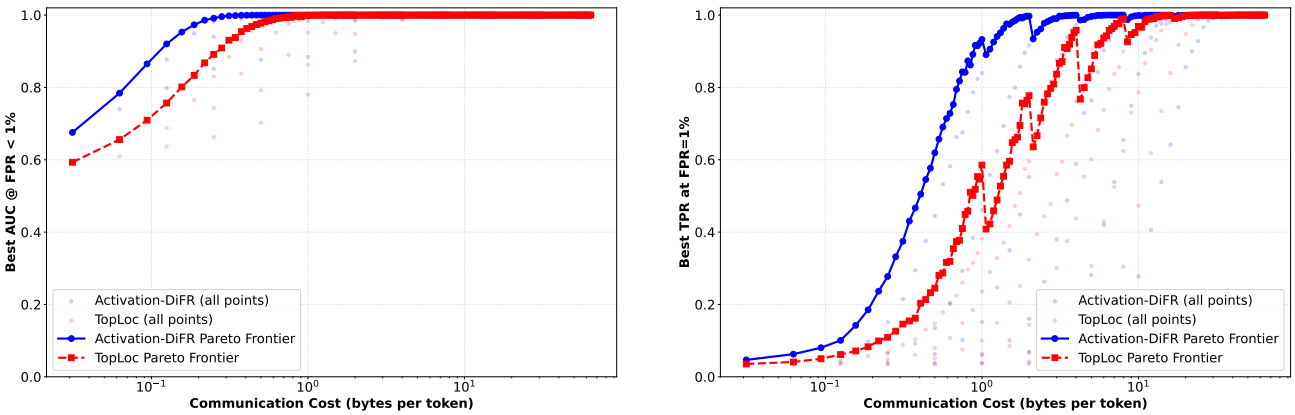
J.3. Threshold-Based Cost Comparison

To quantify the communication savings more precisely, we measure the minimum cost required to achieve specific AUC thresholds: 0.95, 0.99, 0.999, and 0.9999. For each threshold τ and each method (Activation-DiFR or TOPLOC), we identify the lowest communication cost at which AUC $\geq \tau$. Figure 9 presents these costs as bar charts, with communication costs annotated above each threshold. Across all three models and all thresholds examined, Activation-DiFR achieves the target AUC at 25–75% lower communication cost compared to TOPLOC.



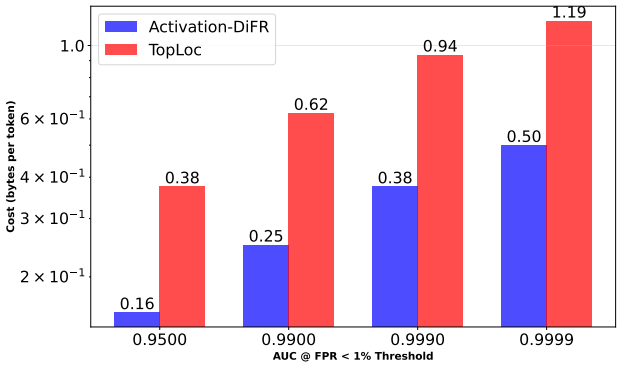
(a) Qwen3-8B *(b)* Llama-3.1-8B-Instruct

Figure 7. **Pareto frontiers for communication cost vs. detection accuracy for Qwen3-8B and Llama 3.1 8B.** Each point represents a (k, B) configuration's communication cost (x-axis, log scale) and AUC at FPR=1%. Solid blue lines show Activation DiFR's Pareto frontier; dashed red lines show TOPLOC's frontier. Activation DiFR dominates TOPLOC across all cost levels and model architectures on the fp8 KV cache quantization detection task.

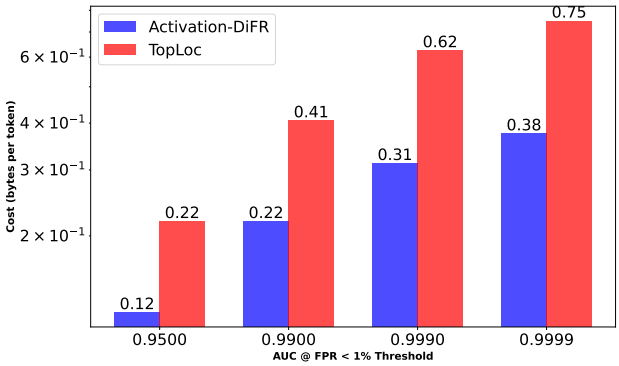


(a) Qwen3-30B-A3B (Matched H200 GPUs) *(b)* Qwen3-30B-A3B (A100 vs H200 GPU)

Figure 8. **Pareto frontiers for communication cost vs. detection accuracy for Qwen3-30B-A3B, with matching GPUs between provider and verifier (H200s) and mismatched GPUs (A100 vs H200).** Each point represents a (k, B) configuration's communication cost (x-axis, log scale) and AUC at FPR=1%. Solid blue lines show Activation DiFR's Pareto frontier; dashed red lines show TOPLOC's frontier. Activation DiFR dominates TOPLOC across all cost levels and model architectures on the fp8 KV cache quantization detection task.

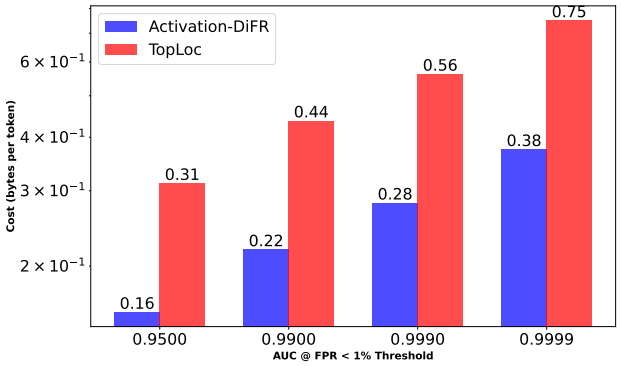


(a) Qwen3-8B

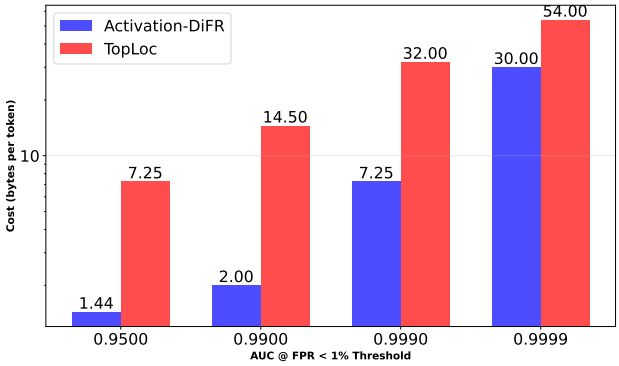


(b) Llama-3.1-8B-Instruct

Figure 9. **Communication cost required to achieve AUC thresholds for Llama 3.1 8B and Qwen3-8B.** For each AUC threshold (x-axis) and method, we report the minimum communication cost (y-axis) required to achieve at least that AUC at FPR=1%. Percentage labels indicate Activation DiFR's relative cost reduction compared to TOPLOC. Activation DiFR consistently achieves target AUC at 25–50% lower communication overhead across all models and thresholds on the fp8 KV cache quantization detection task.



(a) Qwen3-30B-A3B (Matched H200 GPUs)



(b) Qwen3-30B-A3B (A100 vs H200 GPU)

Figure 10. **Pareto frontiers for communication cost vs. detection accuracy for Qwen3-30B-A3B, with matching GPUs between provider and verifier (H200s) and mismatched GPUs (A100 vs H200).** For each AUC threshold (x-axis) and method, we report the minimum communication cost (y-axis) required to achieve at least that AUC at FPR=1%. Percentage labels indicate Activation DiFR's relative cost reduction compared to TOPLOC. Activation DiFR consistently achieves target AUC at 25–50% lower communication overhead across all models and thresholds on the fp8 KV cache quantization detection task.

K. Misconfiguration Detection AUC Results

We include both AUC and AUC @ FPR $< 1\%$ for our quantization and incorrect sampling detection with Qwen3-30B-A3B, which is our most challenging model. These values underlie the abstract's 'AUC > 0.999' claims.

Batch Size	Incorrect Seed		Temperature (1.0→1.1)		4-bit Model Quant.		FP8 KV Cache Quant.	
	AUC	AUC < FPR 1%	AUC	AUC < FPR 1%	AUC	AUC < FPR 1%	AUC	AUC < FPR 1%
1	0.5694	0.5314	0.5006	0.5000	0.5284	0.5124	0.5111	0.5045
3	0.6762	0.5493	0.5016	0.5000	0.5802	0.5160	0.5320	0.5051
10	0.8708	0.6737	0.5048	0.5000	0.7065	0.5474	0.5930	0.5105
30	0.9834	0.8668	0.5149	0.5000	0.8668	0.6067	0.6947	0.5178
100	0.9999	0.9976	0.5284	0.4994	0.9776	0.7915	0.8317	0.5372
300	1.0000	1.0000	0.5561	0.4977	0.9993	0.9768	0.9218	0.5866
1000	1.0000	1.0000	0.6235	0.4975	1.0000	1.0000	0.9786	0.7177
3000	1.0000	1.0000	0.7016	0.4975	1.0000	1.0000	0.9976	0.9302
10000	1.0000	1.0000	0.7688	0.4975	1.0000	1.0000	1.0000	1.0000

Table 2. AUC values for Token-DiFR score across different batch sizes and incorrect configurations for Qwen-3-30B-A3B.

Batch Size	Incorrect Seed		Temperature (1.0→1.1)		4-bit Model Quant.		FP8 KV Cache Quant.	
	AUC	AUC < FPR 1%	AUC	AUC < FPR 1%	AUC	AUC < FPR 1%	AUC	AUC < FPR 1%
1	0.8475	0.5075	0.8521	0.5076	0.9911	0.6658	0.9206	0.5115
2	0.8577	0.5074	0.8621	0.5076	0.9997	0.9854	0.9507	0.5706
4	0.8663	0.5072	0.8707	0.5075	1.0000	1.0000	0.9836	0.6999
8	0.8693	0.5068	0.8751	0.5075	1.0000	1.0000	0.9982	0.9294
16	0.8669	0.5060	0.8756	0.5074	1.0000	1.0000	1.0000	0.9984

Table 3. AUC values for Activation-DiFR score across different batch sizes and incorrect configurations for Qwen-3-30B-A3B.

L. Raw Score Distributions

This appendix provides comprehensive statistics of the raw score distributions for each model, configuration, GPU type, and metric. The tables report the count of samples, share of infinite values (where applicable), mean, standard deviation, and key percentiles (90th, 99th, 99.9th, 99.99th, and 99.999th). These statistics characterize the verification score distributions under both correct and incorrect configurations, providing insight into the separability of different misconfiguration types. All metrics are defined in Section 3.3.

Token-DiFR (margin) raw score distributions

Model	Config	GPU	Count	∞	Mean	Std	p90	p99	p99.9	p99.99	p99.999
Qwen3-30B-A3B	reference	a100	629294	0.0011	0.02	0.25	0.00	0.61	4.02	6.53	9.13
Qwen3-30B-A3B	reference	h200	627756	0.0001	0.00	0.02	0.00	0.00	0.01	0.91	2.42
Qwen3-30B-A3B	correct_tp4	a100	629072	0.0012	0.00	0.06	0.00	0.10	0.75	2.58	4.97
Qwen3-30B-A3B	correct_tp4	h200	630195	0.0011	0.00	0.06	0.00	0.10	0.72	2.47	4.75
Qwen3-30B-A3B	correct_hf	h200	806271	0.0024	0.06	0.41	0.00	2.26	5.24	7.48	9.52
Qwen3-30B-A3B	incorrect_adv_kv_fp8_temp_0_96	h200	604449	0.0031	0.02	0.14	0.00	0.56	1.97	3.94	5.89
Qwen3-30B-A3B	incorrect_config_random_bug_k_2	h200	625908	0.0031	0.00	0.12	0.00	0.00	1.86	5.03	7.64
Qwen3-30B-A3B	incorrect_config_random_bug_k_32	h200	621929	0.0094	0.00	0.06	0.00	0.00	0.12	3.15	6.79
Qwen3-30B-A3B	incorrect_kv_cache_quant_fp8	h200	603591	0.0040	0.02	0.13	0.00	0.52	1.80	3.62	6.04
Qwen3-30B-A3B	incorrect_model_quant_4bit	h200	559981	0.0114	0.05	0.26	0.00	1.35	3.07	4.88	7.88
Qwen3-30B-A3B	incorrect_model_quant_fp8	h200	661475	0.0027	0.01	0.11	0.00	0.41	1.58	3.47	5.12
Qwen3-30B-A3B	incorrect_seed_43	h200	665261	0.0001	0.27	0.85	0.93	4.27	6.70	9.05	11.19
Qwen3-30B-A3B	incorrect_temperature_1_1	h200	615796	0.0040	0.00	0.02	0.00	0.00	0.21	0.51	1.42
Qwen3-30B-A3B	incorrect_top_p_0_85	h200	628161	0.0000	0.02	0.21	0.00	0.76	3.13	5.45	8.10
Qwen3-8B	reference	a100	786831	0.0005	0.00	0.04	0.00	0.00	0.22	2.05	4.49
Qwen3-8B	reference	h200	784915	0.0005	0.00	0.04	0.00	0.00	0.21	1.89	4.28
Qwen3-8B	correct_tp4	a100	788186	0.0006	0.00	0.04	0.00	0.00	0.23	1.94	4.28
Qwen3-8B	correct_tp4	h200	785547	0.0005	0.00	0.04	0.00	0.00	0.23	1.97	4.50
Qwen3-8B	correct_hf	h200	783799	0.0006	0.00	0.04	0.00	0.00	0.21	1.82	4.38
Qwen3-8B	incorrect_adv_kv_fp8_temp_0_96	h200	774505	0.0018	0.01	0.11	0.00	0.37	1.63	3.83	5.75
Qwen3-8B	incorrect_adv_model_4bit_temp_0_89	h200	708037	0.0051	0.06	0.30	0.00	1.55	3.53	5.54	7.59
Qwen3-8B	incorrect_config_random_bug_k_2	h200	783008	0.0029	0.01	0.14	0.00	0.01	2.37	5.13	7.47
Qwen3-8B	incorrect_config_random_bug_k_32	h200	777720	0.0097	0.00	0.08	0.00	0.00	0.28	4.28	6.84
Qwen3-8B	incorrect_kv_cache_quant_fp8	h200	764520	0.0028	0.01	0.10	0.00	0.34	1.33	3.58	5.80
Qwen3-8B	incorrect_model_quant_4bit	h200	710621	0.0085	0.05	0.27	0.00	1.36	3.11	5.25	7.09
Qwen3-8B	incorrect_model_quant_fp8	h200	778203	0.0021	0.01	0.09	0.00	0.30	1.24	3.38	5.53
Qwen3-8B	incorrect_seed_43	h200	738399	0.0005	0.38	1.01	1.56	4.77	7.29	9.71	11.87
Qwen3-8B	incorrect_temperature_1_1	h200	781435	0.0059	0.00	0.02	0.00	0.06	0.28	0.43	1.28
Qwen3-8B	incorrect_top_p_0_85	h200	783913	0.0000	0.03	0.24	0.00	1.04	3.36	5.64	8.35
Llama-3.1-8B	reference	a100	398030	0.0004	0.00	0.04	0.00	0.00	0.12	1.77	4.33
Llama-3.1-8B	reference	h200	400206	0.0004	0.00	0.04	0.00	0.00	0.12	1.93	4.28
Llama-3.1-8B	correct_tp4	a100	401402	0.0005	0.00	0.04	0.00	0.00	0.15	1.91	4.53
Llama-3.1-8B	correct_tp4	h200	402934	0.0005	0.00	0.04	0.00	0.00	0.14	1.83	3.97
Llama-3.1-8B	correct_hf	h200	398700	0.0005	0.00	0.03	0.00	0.00	0.12	1.64	3.57
Llama-3.1-8B	incorrect_adv_kv_fp8_temp_0_96	h200	399515	0.0016	0.01	0.10	0.00	0.28	1.48	3.47	5.05
Llama-3.1-8B	incorrect_adv_model_4bit_temp_0_89	h200	382343	0.0039	0.05	0.25	0.00	1.23	3.12	5.32	7.28
Llama-3.1-8B	incorrect_config_random_bug_k_2	h200	399210	0.0028	0.01	0.14	0.00	0.00	2.45	5.11	6.62
Llama-3.1-8B	incorrect_config_random_bug_k_32	h200	396691	0.0091	0.00	0.13	0.00	0.00	1.39	6.27	7.57
Llama-3.1-8B	incorrect_kv_cache_quant_fp8	h200	399409	0.0029	0.01	0.08	0.00	0.24	1.11	3.01	4.91
Llama-3.1-8B	incorrect_model_quant_4bit	h200	381999	0.0092	0.04	0.21	0.00	1.04	2.69	4.55	5.91
Llama-3.1-8B	incorrect_model_quant_fp8	h200	399564	0.0025	0.01	0.08	0.00	0.24	1.03	3.24	5.33
Llama-3.1-8B	incorrect_seed_43	h200	392316	0.0004	0.51	1.23	2.16	5.58	8.02	10.23	11.84
Llama-3.1-8B	incorrect_temperature_1_1	h200	388448	0.0085	0.00	0.02	0.00	0.09	0.30	0.45	1.45
Llama-3.1-8B	incorrect_top_p_0_85	h200	408071	0.0000	0.03	0.25	0.00	1.12	3.48	5.75	8.27

Table 4. Summary statistics for the sampled token margin between the verifier's most probable token and the claimed token. The ∞ column indicates the fraction of tokens where the claimed token was filtered out by top-k or top-p sampling.

Cross-entropy raw score distributions

Model	Config	GPU	Count	∞	Mean	Std	p90	p99	p99.9	p99.99	p99.999
Qwen3-30B-A3B	reference	a100	629294	0.0011	0.27	0.58	0.97	2.81	3.94	4.97	5.45
Qwen3-30B-A3B	reference	h200	627756	0.0001	0.27	0.59	0.97	2.81	3.97	5.05	5.50
Qwen3-30B-A3B	correct_tp4	a100	629072	0.0012	0.27	0.59	0.97	2.81	3.98	4.96	5.41
Qwen3-30B-A3B	correct_tp4	h200	630195	0.0011	0.27	0.58	0.97	2.81	3.93	4.89	5.38
Qwen3-30B-A3B	correct_hf	h200	806271	0.0024	0.27	0.59	0.96	2.81	4.06	5.09	5.50
Qwen3-30B-A3B	incorrect_adv_kv_fp8_temp_0_96	h200	604449	0.0031	0.27	0.58	0.97	2.79	3.89	4.80	5.40
Qwen3-30B-A3B	incorrect_config_random_bug_k_2	h200	625908	0.0031	0.28	0.59	0.97	2.81	3.98	5.05	5.50
Qwen3-30B-A3B	incorrect_config_random_bug_k_32	h200	621929	0.0094	0.28	0.59	0.97	2.81	4.00	5.07	5.52
Qwen3-30B-A3B	incorrect_kv_cache_quant_fp8	h200	603591	0.0040	0.28	0.60	0.97	2.83	3.98	4.97	5.47
Qwen3-30B-A3B	incorrect_model_quant_4bit	h200	559981	0.0114	0.30	0.64	1.12	2.97	4.09	5.08	5.51
Qwen3-30B-A3B	incorrect_model_quant_fp8	h200	661475	0.0027	0.27	0.59	0.97	2.81	3.92	4.88	5.45
Qwen3-30B-A3B	incorrect_seed_43	h200	665261	0.0001	0.27	0.59	0.97	2.81	4.01	4.99	5.52
Qwen3-30B-A3B	incorrect_temperature_1_1	h200	615796	0.0040	0.29	0.62	1.02	2.94	4.14	5.09	5.46
Qwen3-30B-A3B	incorrect_top_p_0_85	h200	628161	0.0000	0.23	0.46	0.83	2.12	3.01	3.92	4.40
Qwen3-8B	reference	a100	786831	0.0005	0.38	0.69	1.30	3.17	4.35	5.21	5.53
Qwen3-8B	reference	h200	784915	0.0005	0.38	0.69	1.30	3.17	4.37	5.23	5.57
Qwen3-8B	correct_tp4	a100	788186	0.0006	0.38	0.70	1.30	3.17	4.36	5.15	5.52
Qwen3-8B	correct_tp4	h200	785547	0.0005	0.37	0.69	1.29	3.17	4.33	5.15	5.52
Qwen3-8B	correct_hf	h200	783799	0.0006	0.38	0.70	1.30	3.17	4.36	5.23	5.54
Qwen3-8B	incorrect_adv_kv_fp8_temp_0_96	h200	774505	0.0018	0.37	0.68	1.26	3.08	4.26	5.05	5.41
Qwen3-8B	incorrect_adv_model_4bit_temp_0_89	h200	708037	0.0051	0.37	0.67	1.26	3.05	4.20	5.05	5.55
Qwen3-8B	incorrect_config_random_bug_k_2	h200	783008	0.0029	0.38	0.70	1.31	3.17	4.37	5.23	5.57
Qwen3-8B	incorrect_config_random_bug_k_32	h200	777720	0.0097	0.38	0.70	1.30	3.18	4.39	5.23	5.57
Qwen3-8B	incorrect_kv_cache_quant_fp8	h200	764520	0.0028	0.38	0.70	1.31	3.17	4.33	5.18	5.53
Qwen3-8B	incorrect_model_quant_4bit	h200	710621	0.0085	0.40	0.72	1.37	3.24	4.42	5.17	5.55
Qwen3-8B	incorrect_model_quant_fp8	h200	778203	0.0021	0.38	0.69	1.30	3.14	4.34	5.22	5.51
Qwen3-8B	incorrect_seed_43	h200	738399	0.0005	0.38	0.70	1.31	3.20	4.44	5.25	5.58
Qwen3-8B	incorrect_temperature_1_1	h200	781435	0.0059	0.40	0.74	1.41	3.32	4.50	5.25	5.59
Qwen3-8B	incorrect_top_p_0_85	h200	783913	0.0000	0.31	0.55	1.09	2.43	3.38	4.20	4.63
Llama-3.1-8B	reference	a100	398030	0.0004	0.51	0.93	1.80	4.22	5.26	5.60	5.74
Llama-3.1-8B	reference	h200	400206	0.0004	0.50	0.92	1.77	4.22	5.28	5.61	5.76
Llama-3.1-8B	correct_tp4	a100	401402	0.0005	0.51	0.93	1.80	4.23	5.28	5.61	5.76
Llama-3.1-8B	correct_tp4	h200	402934	0.0005	0.51	0.93	1.80	4.24	5.27	5.61	5.79
Llama-3.1-8B	correct_hf	h200	398700	0.0005	0.51	0.93	1.79	4.23	5.29	5.61	5.81
Llama-3.1-8B	incorrect_adv_kv_fp8_temp_0_96	h200	399515	0.0016	0.48	0.89	1.70	4.05	5.15	5.54	5.75
Llama-3.1-8B	incorrect_adv_model_4bit_temp_0_89	h200	382343	0.0039	0.46	0.85	1.61	3.86	5.06	5.50	5.68
Llama-3.1-8B	incorrect_config_random_bug_k_2	h200	399210	0.0028	0.50	0.93	1.78	4.21	5.28	5.61	5.76
Llama-3.1-8B	incorrect_config_random_bug_k_32	h200	396691	0.0091	0.51	0.93	1.78	4.25	5.30	5.62	5.76
Llama-3.1-8B	incorrect_kv_cache_quant_fp8	h200	399409	0.0029	0.53	0.95	1.86	4.27	5.28	5.62	5.79
Llama-3.1-8B	incorrect_model_quant_4bit	h200	381999	0.0092	0.56	0.98	1.98	4.34	5.29	5.61	5.81
Llama-3.1-8B	incorrect_model_quant_fp8	h200	399564	0.0025	0.52	0.94	1.82	4.24	5.26	5.61	5.76
Llama-3.1-8B	incorrect_seed_43	h200	392316	0.0004	0.51	0.94	1.80	4.28	5.29	5.63	5.83
Llama-3.1-8B	incorrect_temperature_1_1	h200	388448	0.0085	0.62	1.07	2.21	4.61	5.42	5.70	5.89
Llama-3.1-8B	incorrect_top_p_0_85	h200	408071	0.0000	0.37	0.70	1.34	3.21	4.25	4.64	4.82

Table 5. Summary statistics for the negative log-likelihood of the claimed token under the verifier's softmax distribution. The ∞ column indicates the fraction of tokens where either the probability underflowed to 0 before taking the log, or the token was filtered out by top-k or top-p sampling.

Token-DiFR (exact match) raw score distributions

Model	Config	GPU	Count	Mean	Std	p90	p99	p99.9	p99.99	p99.999
Qwen3-30B-A3B	reference	a100	630001	0.97	0.16	1.00	1.00	1.00	1.00	1.00
Qwen3-30B-A3B	reference	h200	627824	1.00	0.03	1.00	1.00	1.00	1.00	1.00
Qwen3-30B-A3B	correct_tp4	a100	629846	0.98	0.13	1.00	1.00	1.00	1.00	1.00
Qwen3-30B-A3B	correct_tp4	h200	630916	0.98	0.13	1.00	1.00	1.00	1.00	1.00
Qwen3-30B-A3B	correct_hf	h200	808241	0.95	0.21	1.00	1.00	1.00	1.00	1.00
Qwen3-30B-A3B	incorrect_adv_kv_fp8_temp_0_96	h200	606314	0.96	0.19	1.00	1.00	1.00	1.00	1.00
Qwen3-30B-A3B	incorrect_config_random_bug_k_2	h200	627824	0.99	0.08	1.00	1.00	1.00	1.00	1.00
Qwen3-30B-A3B	incorrect_config_random_bug_k_32	h200	627824	0.99	0.10	1.00	1.00	1.00	1.00	1.00
Qwen3-30B-A3B	incorrect_kv_cache_quant_fp8	h200	606023	0.96	0.19	1.00	1.00	1.00	1.00	1.00
Qwen3-30B-A3B	incorrect_model_quant_4bit	h200	566434	0.93	0.26	1.00	1.00	1.00	1.00	1.00
Qwen3-30B-A3B	incorrect_model_quant_fp8	h200	663284	0.97	0.17	1.00	1.00	1.00	1.00	1.00
Qwen3-30B-A3B	incorrect_seed_43	h200	665315	0.85	0.36	1.00	1.00	1.00	1.00	1.00
Qwen3-30B-A3B	incorrect_temperature_1_1	h200	618265	0.99	0.12	1.00	1.00	1.00	1.00	1.00
Qwen3-30B-A3B	incorrect_top_p_0_85	h200	628170	0.98	0.14	1.00	1.00	1.00	1.00	1.00
Qwen3-8B	reference	a100	787234	0.99	0.09	1.00	1.00	1.00	1.00	1.00
Qwen3-8B	reference	h200	785304	0.99	0.09	1.00	1.00	1.00	1.00	1.00
Qwen3-8B	correct_tp4	a100	788648	0.99	0.09	1.00	1.00	1.00	1.00	1.00
Qwen3-8B	correct_tp4	h200	785979	0.99	0.09	1.00	1.00	1.00	1.00	1.00
Qwen3-8B	correct_hf	h200	784280	0.99	0.09	1.00	1.00	1.00	1.00	1.00
Qwen3-8B	incorrect_adv_kv_fp8_temp_0_96	h200	775897	0.97	0.18	1.00	1.00	1.00	1.00	1.00
Qwen3-8B	incorrect_adv_model_4bit_temp_0_89	h200	711666	0.92	0.28	1.00	1.00	1.00	1.00	1.00
Qwen3-8B	incorrect_config_random_bug_k_2	h200	785304	0.99	0.11	1.00	1.00	1.00	1.00	1.00
Qwen3-8B	incorrect_config_random_bug_k_32	h200	785304	0.98	0.13	1.00	1.00	1.00	1.00	1.00
Qwen3-8B	incorrect_kv_cache_quant_fp8	h200	766667	0.97	0.18	1.00	1.00	1.00	1.00	1.00
Qwen3-8B	incorrect_model_quant_4bit	h200	716701	0.92	0.27	1.00	1.00	1.00	1.00	1.00
Qwen3-8B	incorrect_model_quant_fp8	h200	779878	0.97	0.17	1.00	1.00	1.00	1.00	1.00
Qwen3-8B	incorrect_seed_43	h200	738774	0.79	0.40	1.00	1.00	1.00	1.00	1.00
Qwen3-8B	incorrect_temperature_1_1	h200	786035	0.98	0.14	1.00	1.00	1.00	1.00	1.00
Qwen3-8B	incorrect_top_p_0_85	h200	783913	0.97	0.18	1.00	1.00	1.00	1.00	1.00
Llama-3.1-8B	reference	a100	398185	0.99	0.08	1.00	1.00	1.00	1.00	1.00
Llama-3.1-8B	reference	h200	400348	0.99	0.08	1.00	1.00	1.00	1.00	1.00
Llama-3.1-8B	correct_tp4	a100	401598	0.99	0.09	1.00	1.00	1.00	1.00	1.00
Llama-3.1-8B	correct_tp4	h200	403141	0.99	0.09	1.00	1.00	1.00	1.00	1.00
Llama-3.1-8B	correct_hf	h200	398902	0.99	0.08	1.00	1.00	1.00	1.00	1.00
Llama-3.1-8B	incorrect_adv_kv_fp8_temp_0_96	h200	400168	0.97	0.18	1.00	1.00	1.00	1.00	1.00
Llama-3.1-8B	incorrect_adv_model_4bit_temp_0_89	h200	383836	0.93	0.26	1.00	1.00	1.00	1.00	1.00
Llama-3.1-8B	incorrect_config_random_bug_k_2	h200	400348	0.99	0.11	1.00	1.00	1.00	1.00	1.00
Llama-3.1-8B	incorrect_config_random_bug_k_32	h200	400348	0.98	0.13	1.00	1.00	1.00	1.00	1.00
Llama-3.1-8B	incorrect_kv_cache_quant_fp8	h200	400573	0.97	0.17	1.00	1.00	1.00	1.00	1.00
Llama-3.1-8B	incorrect_model_quant_4bit	h200	385544	0.92	0.26	1.00	1.00	1.00	1.00	1.00
Llama-3.1-8B	incorrect_model_quant_fp8	h200	400549	0.97	0.17	1.00	1.00	1.00	1.00	1.00
Llama-3.1-8B	incorrect_seed_43	h200	392476	0.76	0.43	1.00	1.00	1.00	1.00	1.00
Llama-3.1-8B	incorrect_temperature_1_1	h200	391774	0.97	0.16	1.00	1.00	1.00	1.00	1.00
Llama-3.1-8B	incorrect_top_p_0_85	h200	408071	0.97	0.18	1.00	1.00	1.00	1.00	1.00

Table 6. Summary statistics for the binary exact-match rate: the fraction of tokens where the provider and verifier sampled the same token under a shared seed. Since this is a binary metric (0 or 1), there are no infinite values.

Token-DiFR (likelihood, $\sigma = 0.02$) raw score distributions

Model	Config	GPU	Count	∞	Mean	Std	p90	p99	p99.9	p99.99	p99.999
Qwen3-30B-A3B	reference	a100	629294	0.0011	40.96	710.36	0.00	237.01	10127.86	26636.31	52152.83
Qwen3-30B-A3B	reference	h200	627756	0.0001	0.27	38.23	0.00	0.00	0.99	522.25	3672.64
Qwen3-30B-A3B	correct_tp4	a100	629072	0.0012	2.48	102.66	0.00	8.91	356.07	4155.61	15459.88
Qwen3-30B-A3B	correct_tp4	h200	630195	0.0011	2.28	105.55	0.00	8.39	327.01	3809.94	14106.97
Qwen3-30B-A3B	correct_hf	h200	806271	0.0024	109.36	1133.59	0.00	3203.67	17191.44	35016.00	56656.67
Qwen3-30B-A3B	incorrect_adv_kv_fp8_temp_0_96	h200	604449	0.0031	12.55	232.32	0.00	198.55	2420.33	9693.36	21700.69
Qwen3-30B-A3B	incorrect_config_random_bug_k_2	h200	625908	0.0031	8.76	329.24	0.00	0.00	2159.98	15818.20	36520.42
Qwen3-30B-A3B	incorrect_config_random_bug_k_32	h200	621929	0.0094	2.20	195.42	0.00	0.00	10.77	6225.48	28781.38
Qwen3-30B-A3B	incorrect_kv_cache_quant_fp8	h200	603591	0.0040	10.87	220.33	0.00	172.01	2039.49	8209.02	22831.76
Qwen3-30B-A3B	incorrect_model_quant_4bit	h200	559981	0.0114	43.55	447.41	0.00	1146.94	5897.47	14895.77	38808.91
Qwen3-30B-A3B	incorrect_model_quant_fp8	h200	661475	0.0027	8.04	168.83	0.00	108.13	1564.10	7550.36	16418.96
Qwen3-30B-A3B	incorrect_seed_43	h200	665261	0.0001	498.82	2353.89	547.69	11376.17	28051.18	51245.66	78249.20
Qwen3-30B-A3B	incorrect_temperature_1_1	h200	615796	0.0040	0.19	12.25	0.00	0.00	30.15	168.89	1268.61
Qwen3-30B-A3B	incorrect_top_p_0_85	h200	628161	0.0000	27.66	468.32	0.00	362.01	6146.39	18599.42	41019.43
Qwen3-8B	reference	a100	786831	0.0005	1.05	94.81	0.00	0.00	34.12	2628.20	12586.91
Qwen3-8B	reference	h200	784915	0.0005	0.92	73.40	0.00	0.00	30.30	2233.97	11444.36
Qwen3-8B	correct_tp4	a100	788186	0.0006	0.99	75.30	0.00	0.00	35.06	2358.73	11468.82
Qwen3-8B	correct_tp4	h200	785547	0.0005	1.10	87.74	0.00	0.00	35.17	2440.59	12695.14
Qwen3-8B	correct_hf	h200	783799	0.0006	0.85	76.01	0.00	0.00	30.88	2077.44	11973.31
Qwen3-8B	incorrect_adv_kv_fp8_temp_0_96	h200	774505	0.0018	8.32	203.76	0.00	89.68	1663.93	9181.77	20706.42
Qwen3-8B	incorrect_adv_model_4bit_temp_0_89	h200	708037	0.0051	59.88	554.57	0.00	1500.82	7794.26	19175.61	36001.17
Qwen3-8B	incorrect_config_random_bug_k_2	h200	783008	0.0029	11.68	382.62	0.00	0.97	3519.33	16425.68	34912.87
Qwen3-8B	incorrect_config_random_bug_k_32	h200	777720	0.0097	4.01	249.74	0.00	0.00	50.61	11466.60	29261.41
Qwen3-8B	incorrect_kv_cache_quant_fp8	h200	764520	0.0028	6.42	174.56	0.00	74.54	1117.25	8024.11	21036.45
Qwen3-8B	incorrect_model_quant_4bit	h200	710621	0.0085	46.14	462.96	0.00	1155.74	6061.66	17207.19	31459.71
Qwen3-8B	incorrect_model_quant_fp8	h200	778203	0.0021	5.66	164.58	0.00	60.84	962.45	7164.61	19103.23
Qwen3-8B	incorrect_seed_43	h200	738399	0.0005	727.44	2930.33	1519.65	14255.76	33201.36	58969.98	88083.83
Qwen3-8B	incorrect_temperature_1_1	h200	781435	0.0059	0.29	21.52	0.00	3.71	51.78	117.61	1029.91
Qwen3-8B	incorrect_top_p_0_85	h200	783913	0.0000	36.04	531.94	0.00	683.94	7069.84	19913.24	43582.39
Llama-3.1-8B	reference	a100	398030	0.0004	0.89	81.33	0.00	0.00	11.80	1956.56	11729.23
Llama-3.1-8B	reference	h200	400206	0.0004	0.82	78.04	0.00	0.00	11.48	2343.55	11459.91
Llama-3.1-8B	correct_tp4	a100	401402	0.0005	0.94	73.49	0.00	0.00	17.54	2283.42	12815.66
Llama-3.1-8B	correct_tp4	h200	402934	0.0005	0.84	62.90	0.00	0.00	15.22	2108.15	9845.70
Llama-3.1-8B	correct_hf	h200	398700	0.0005	0.60	57.33	0.00	0.00	11.61	1687.17	7990.34
Llama-3.1-8B	incorrect_adv_kv_fp8_temp_0_96	h200	399515	0.0016	6.46	159.49	0.00	53.33	1376.86	7515.03	15938.51
Llama-3.1-8B	incorrect_adv_model_4bit_temp_0_89	h200	382343	0.0039	41.20	459.77	0.00	943.38	6086.98	17721.77	33091.78
Llama-3.1-8B	incorrect_config_random_bug_k_2	h200	399210	0.0028	11.82	347.68	0.00	0.00	3769.77	16308.28	27385.34
Llama-3.1-8B	incorrect_config_random_bug_k_32	h200	396691	0.0091	11.24	458.01	0.00	0.00	1217.70	24609.92	35827.70
Llama-3.1-8B	incorrect_kv_cache_quant_fp8	h200	399409	0.0029	4.38	128.38	0.00	39.21	772.30	5650.39	15063.85
Llama-3.1-8B	incorrect_model_quant_4bit	h200	381999	0.0092	29.73	338.04	0.00	683.93	4522.18	12930.92	21827.17
Llama-3.1-8B	incorrect_model_quant_fp8	h200	399564	0.0025	4.34	146.48	0.00	38.17	673.27	6565.22	17794.56
Llama-3.1-8B	incorrect_seed_43	h200	392316	0.0004	1106.86	3870.31	2927.18	19447.98	40243.01	65425.25	87587.88
Llama-3.1-8B	incorrect_temperature_1_1	h200	388448	0.0085	0.34	10.24	0.00	7.01	59.65	131.60	1324.05
Llama-3.1-8B	incorrect_top_p_0_85	h200	408071	0.0000	39.27	561.00	0.00	794.06	7564.55	20686.86	42736.77

Table 7. Summary statistics for the likelihood-based score derived from the sampled token margin using a Gaussian noise model with standard deviation $\sigma = 0.02$. The ∞ column indicates the fraction of tokens where the claimed token was filtered out by top-k or top-p sampling.

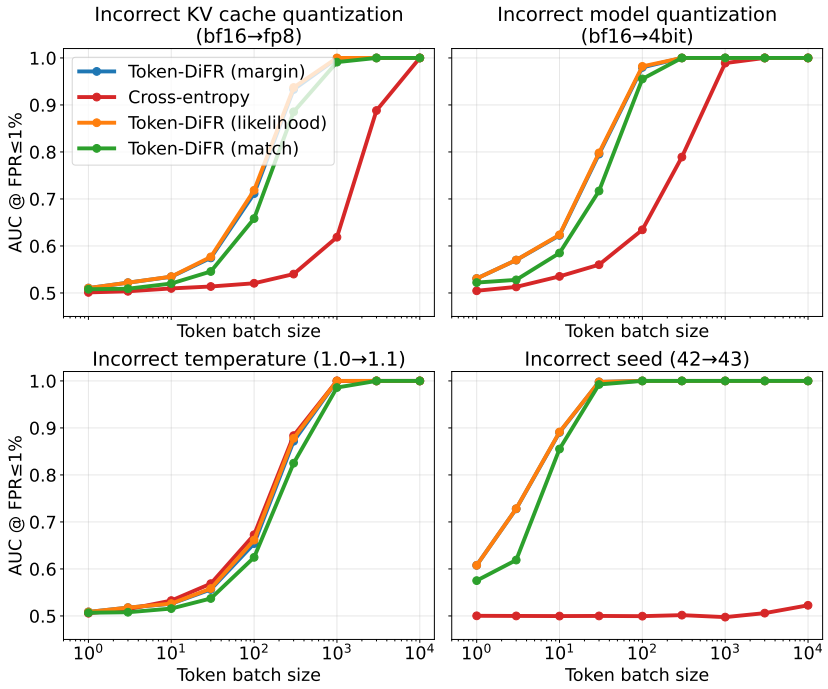
Token-DiFR (likelihood, $\sigma = 0.08$) raw score distributions

Model	Config	GPU	Count	∞	Mean	Std	p90	p99	p99.9	p99.99	p99.999
Qwen3-30B-A3B	reference	a100	629294	0.0011	2.61	44.62	0.00	17.20	637.12	1669.35	3264.44
Qwen3-30B-A3B	reference	h200	627756	0.0001	0.02	2.41	0.00	0.00	0.76	35.39	233.19
Qwen3-30B-A3B	correct_tp4	a100	629072	0.0012	0.18	6.50	0.00	1.71	24.82	263.43	970.57
Qwen3-30B-A3B	correct_tp4	h200	630195	0.0011	0.16	6.67	0.00	1.66	22.97	241.79	885.97
Qwen3-30B-A3B	correct_hf	h200	806271	0.0024	6.95	71.23	0.00	203.82	1078.84	2193.21	3545.97
Qwen3-30B-A3B	incorrect_adv_kv_fp8_temp_0_96	h200	604449	0.0031	0.85	14.70	0.00	14.72	154.73	609.94	1360.77
Qwen3-30B-A3B	incorrect_config_random_bug_k_2	h200	625908	0.0031	0.56	20.68	0.00	0.00	138.40	992.97	2287.25
Qwen3-30B-A3B	incorrect_config_random_bug_k_32	h200	621929	0.0094	0.14	12.26	0.00	0.00	1.88	392.99	1803.45
Qwen3-30B-A3B	incorrect_kv_cache_quant_fp8	h200	603591	0.0040	0.74	13.93	0.00	13.00	130.84	517.09	1431.49
Qwen3-30B-A3B	incorrect_model_quant_4bit	h200	559981	0.0114	2.86	28.29	0.00	74.79	372.46	935.29	2430.31
Qwen3-30B-A3B	incorrect_model_quant_fp8	h200	661475	0.0027	0.55	10.71	0.00	8.80	101.01	475.88	1030.54
Qwen3-30B-A3B	incorrect_seed_43	h200	665261	0.0001	31.64	147.87	37.00	715.19	1757.80	3207.74	4895.66
Qwen3-30B-A3B	incorrect_temperature_1_1	h200	615796	0.0040	0.02	0.81	0.00	0.00	3.42	12.79	82.44
Qwen3-30B-A3B	incorrect_top_p_0_85	h200	628161	0.0000	1.78	29.49	0.00	25.20	388.04	1166.87	2568.49
Qwen3-8B	reference	a100	786831	0.0005	0.08	5.97	0.00	0.00	3.72	167.76	790.91
Qwen3-8B	reference	h200	784915	0.0005	0.07	4.63	0.00	0.00	3.44	143.04	719.45
Qwen3-8B	correct_tp4	a100	788186	0.0006	0.07	4.75	0.00	0.00	3.79	150.86	720.98
Qwen3-8B	correct_tp4	h200	785547	0.0005	0.08	5.53	0.00	0.00	3.79	156.00	797.68
Qwen3-8B	correct_hf	h200	783799	0.0006	0.06	4.79	0.00	0.00	3.48	133.22	752.53
Qwen3-8B	incorrect_adv_kv_fp8_temp_0_96	h200	774505	0.0018	0.57	12.87	0.00	7.57	107.28	577.94	1298.61
Qwen3-8B	incorrect_adv_model_4bit_temp_0_89	h200	708037	0.0051	3.92	35.02	0.00	97.03	491.14	1202.90	2254.79
Qwen3-8B	incorrect_config_random_bug_k_2	h200	783008	0.0029	0.75	24.04	0.00	0.76	223.59	1030.96	2186.76
Qwen3-8B	incorrect_config_random_bug_k_32	h200	777720	0.0097	0.26	15.67	0.00	0.00	4.90	720.84	1833.46
Qwen3-8B	incorrect_kv_cache_quant_fp8	h200	764520	0.0028	0.45	11.03	0.00	6.55	72.92	505.52	1319.25
Qwen3-8B	incorrect_model_quant_4bit	h200	710621	0.0085	3.04	29.26	0.00	75.34	382.74	1079.82	1970.89
Qwen3-8B	incorrect_model_quant_fp8	h200	778203	0.0021	0.40	10.40	0.00	5.61	63.18	451.75	1198.37
Qwen3-8B	incorrect_seed_43	h200	738399	0.0005	46.09	183.99	98.22	895.27	2079.77	3690.57	5510.38
Qwen3-8B	incorrect_temperature_1_1	h200	781435	0.0059	0.04	1.38	0.00	1.17	4.98	9.43	67.43
Qwen3-8B	incorrect_top_p_0_85	h200	783913	0.0000	2.33	33.49	0.00	45.61	445.82	1249.02	2728.71
Llama-3.1-8B	reference	a100	398030	0.0004	0.06	5.12	0.00	0.00	1.97	125.64	737.27
Llama-3.1-8B	reference	h200	400206	0.0004	0.06	4.92	0.00	0.00	1.94	149.91	720.43
Llama-3.1-8B	correct_tp4	a100	401402	0.0005	0.07	4.64	0.00	0.00	2.45	146.14	805.21
Llama-3.1-8B	correct_tp4	h200	402934	0.0005	0.06	3.98	0.00	0.00	2.26	135.15	619.47
Llama-3.1-8B	correct_hf	h200	398700	0.0005	0.04	3.62	0.00	0.00	1.95	108.74	503.41
Llama-3.1-8B	incorrect_adv_kv_fp8_temp_0_96	h200	399515	0.0016	0.45	10.10	0.00	5.09	89.25	473.67	1000.49
Llama-3.1-8B	incorrect_adv_model_4bit_temp_0_89	h200	382343	0.0039	2.72	29.03	0.00	61.98	384.32	1112.00	2072.92
Llama-3.1-8B	incorrect_config_random_bug_k_2	h200	399210	0.0028	0.75	21.86	0.00	0.00	239.27	1023.62	1716.17
Llama-3.1-8B	incorrect_config_random_bug_k_32	h200	396691	0.0091	0.71	28.73	0.00	0.00	79.24	1542.66	2243.95
Llama-3.1-8B	incorrect_kv_cache_quant_fp8	h200	399409	0.0029	0.32	8.13	0.00	4.09	51.19	357.00	945.80
Llama-3.1-8B	incorrect_model_quant_4bit	h200	381999	0.0092	1.99	21.41	0.00	45.61	286.38	812.42	1368.68
Llama-3.1-8B	incorrect_model_quant_fp8	h200	399564	0.0025	0.31	9.25	0.00	4.01	44.94	414.25	1116.55
Llama-3.1-8B	incorrect_seed_43	h200	392316	0.0004	69.94	242.85	186.49	1219.93	2519.96	4094.08	5479.38
Llama-3.1-8B	incorrect_temperature_1_1	h200	388448	0.0085	0.04	0.72	0.00	1.52	5.53	10.35	85.93
Llama-3.1-8B	incorrect_top_p_0_85	h200	408071	0.0000	2.53	35.32	0.00	52.57	476.77	1297.39	2675.85

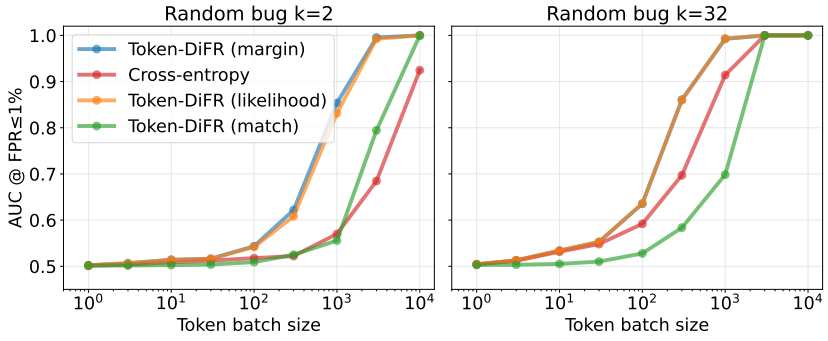
Table 8. Summary statistics for the likelihood-based score derived from the sampled token margin using a Gaussian noise model with standard deviation $\sigma = 0.08$. The ∞ column indicates the fraction of tokens where the claimed token was filtered out by top-k or top-p sampling.

M. All Detection Results Across Models and Configurations

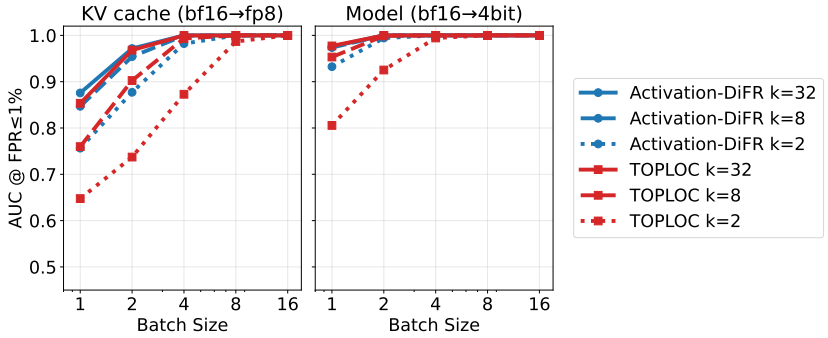
This appendix presents comprehensive batch scaling results across all three evaluated model architectures (Llama-3.1-8B-Instruct, Qwen3-8B, and Qwen3-30B-A3B) and all misconfiguration types. These results demonstrate the robustness of our verification methods across different model architectures and scales.



(a) Standard misconfigurations.

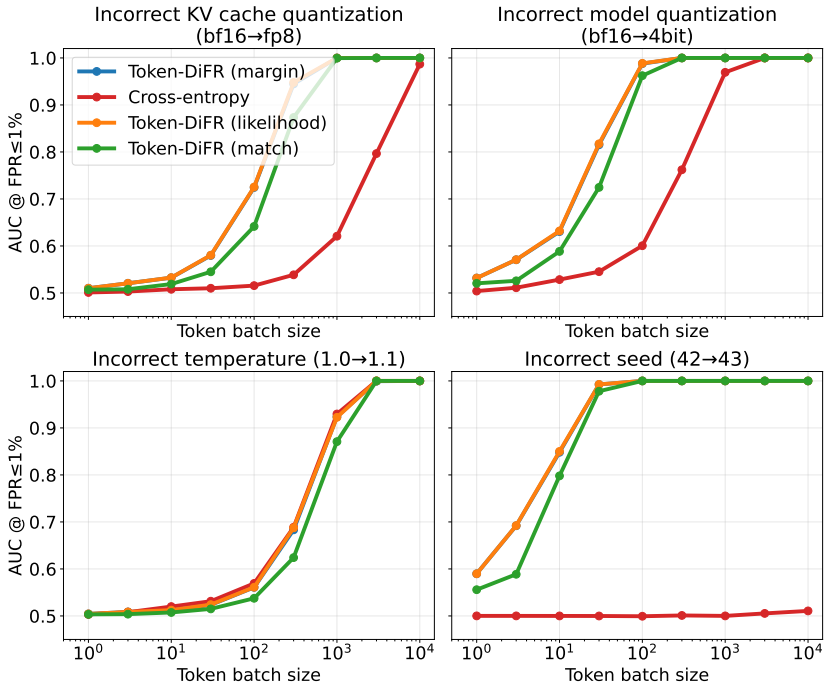


(b) Bug simulations.

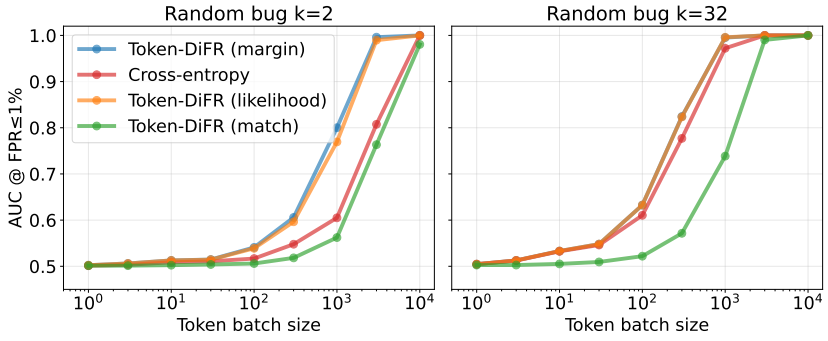


(c) Activation comparisons (AUC).

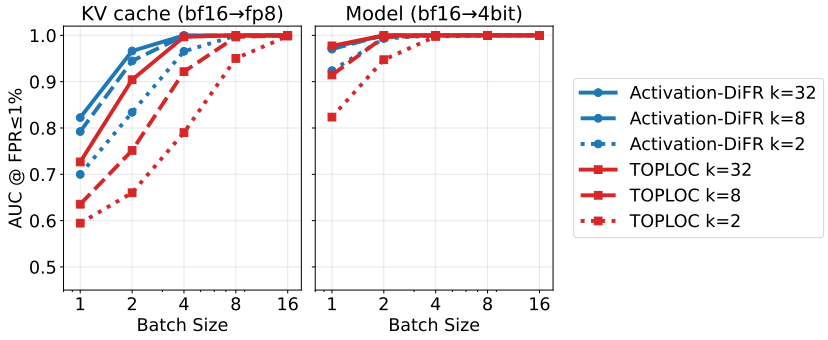
Figure 11. Token-DiFR and Activation-DiFR results for Llama 3.1 8B with the four metric comparison (Token-DiFR, cross entropy, Token-DiFR likelihood, and exact token match).



(a) Standard misconfigurations.



(b) Bug simulations.



(c) Activation comparisons (AUC).

Figure 12. Token-DiFR and Activation-DiFR results for Qwen3-8B with the four metric comparison (Token-DiFR, cross entropy, Token-DiFR likelihood, and exact token match).

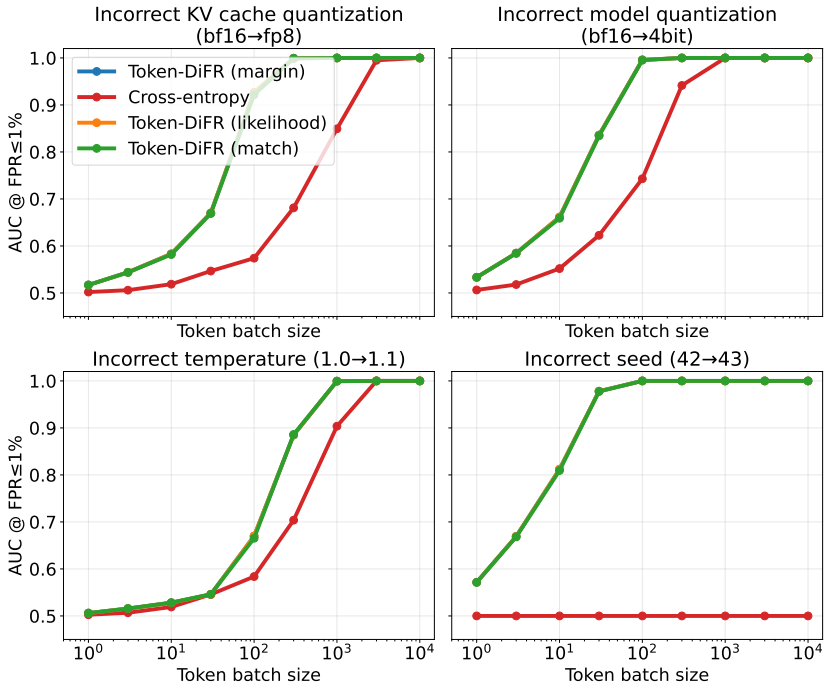
N. Qwen3-30B-A3B Results

For Llama-3.1-8B and Qwen3-8B, we find that detection performance is highly stable across different environments. Varying hardware (A100 vs. H200), tensor-parallel configurations, or inference implementations (vLLM versus HuggingFace) produces only minor changes in the score distributions, and even the most subtle misconfigurations remain reliably detectable using our default Token-DiFR configuration.

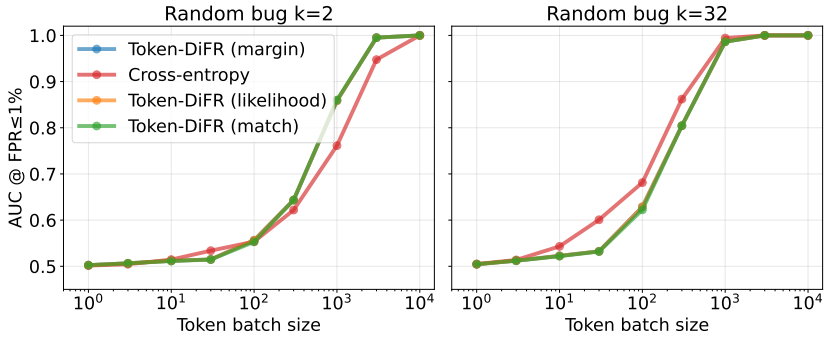
The Qwen3-30B-A3B results exhibit substantially more variability. The largest shifts in behavior arise when switching between vLLM and HuggingFace, although we also observe noticeable changes when moving from H200 to A100 within vLLM. It is unclear whether these differences are driven primarily by the mixture-of-experts architecture or by implementation details in the current inference stacks. This is a particularly challenging setting: when verifier and provider both use our reference H200/vLLM configuration, Qwen3-30B-A3B achieves an unusually high exact-match rate of around 99.9%, while the HuggingFace configuration has a much lower exact-match rate of roughly 95%.

When the verifier matches the provider's setup exactly (reference H200-only), the 30B model behaves much like the smaller models: minor misconfigurations, including small temperature shifts and FP8 KV-cache quantization, are detected accurately and consistently without tuning. As we expand the calibration set to the trusted vLLM pool, which includes multiple parallelism setups and both H100 and A100 runs, performance for the smallest deviations degrades: in particular, we can no longer reliably detect the temperature 1.1 vs. 1.0 misconfiguration at our target false positive rate. Adding HuggingFace runs in the pooled-with-HF configuration broadens the null distribution further, and FP8 KV-cache quantization also becomes difficult to distinguish from the pooled honest baseline. Despite these substantial implementation-driven differences, all configurations still cleanly detect major problems such as 4-bit model quantization and incorrect seeds with modest batch sizes.

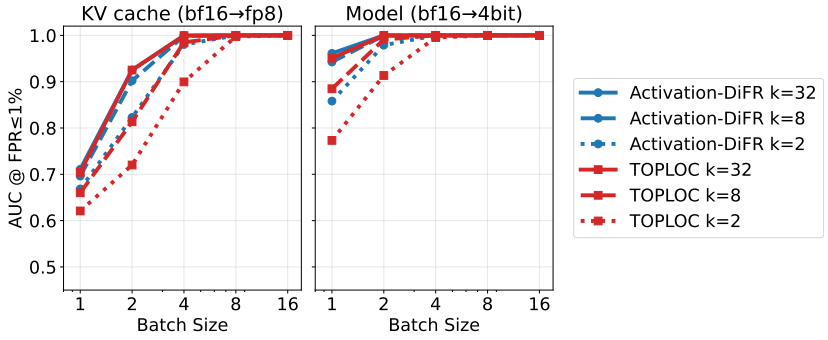
In the following pages, we present Qwen3-30B-A3B results for all three verifier–provider configurations we consider: **reference H200 only**, **trusted vLLM pool**, and **trusted vLLM and HuggingFace pool**.



(a) Standard misconfigurations.

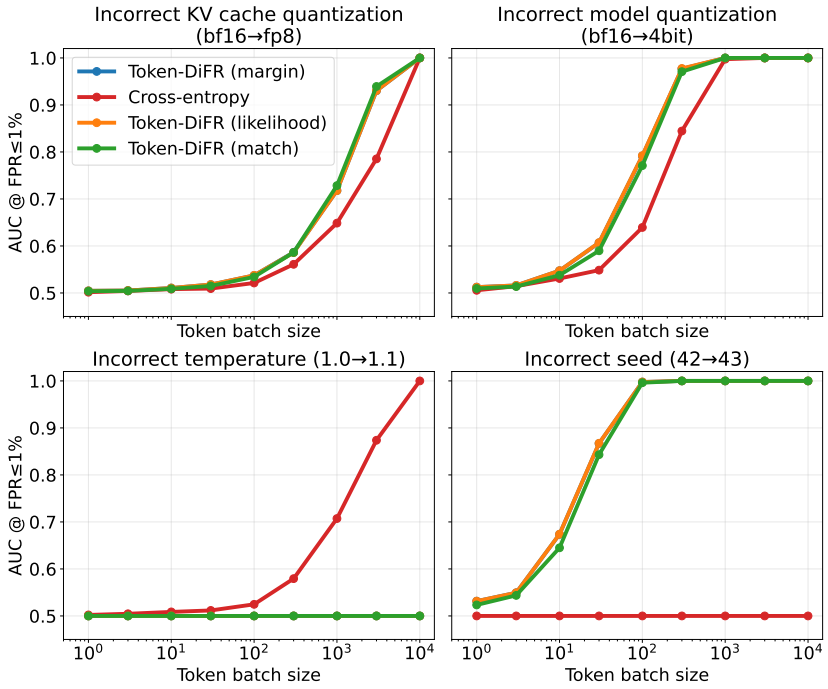


(b) Bug simulations.

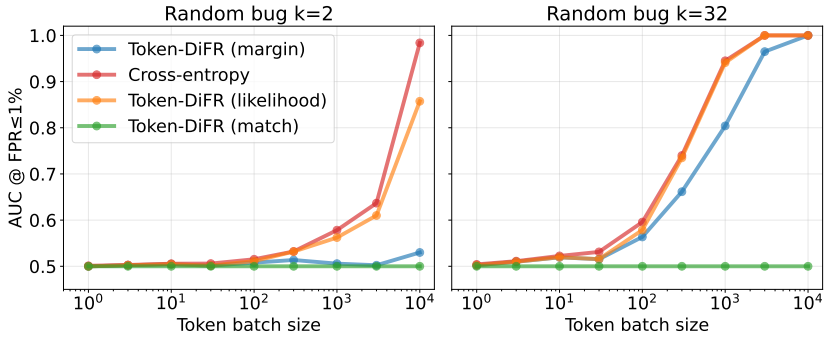


(c) Activation comparisons (AUC).

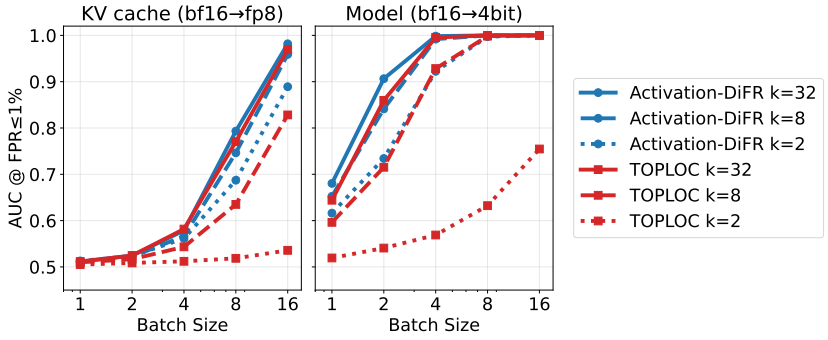
Figure 13. Token-DiFR and Activation-DiFR results for Qwen3-30B-A3B with the reference H200 only setup with the four metric comparison (Token-DiFR, cross entropy, Token-DiFR likelihood, and exact token match).



(a) Standard misconfigurations.

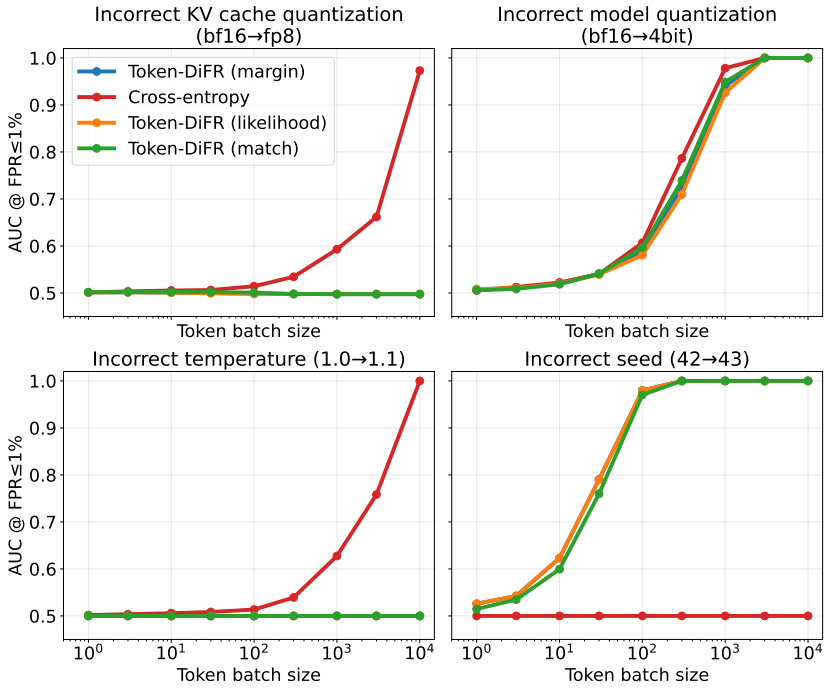


(b) Bug simulations.

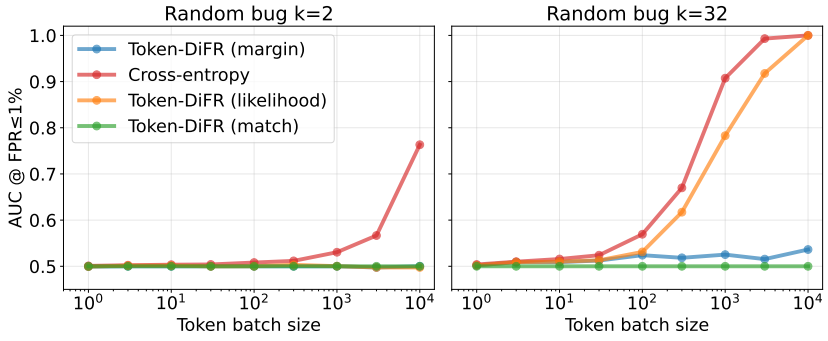


(c) Activation comparisons (AUC).

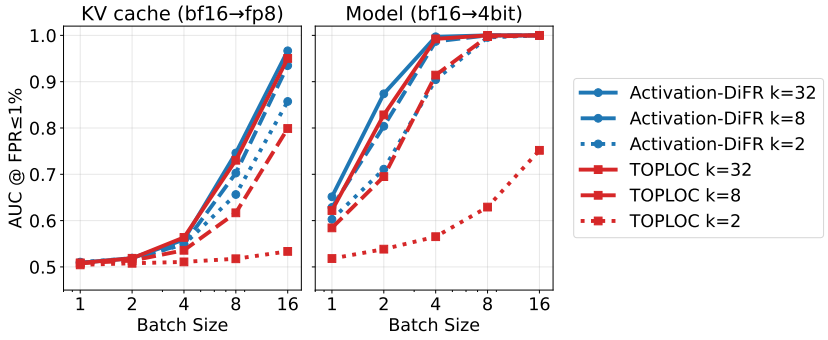
Figure 14. Token-DiFR and Activation-DiFR results for Qwen3-30B-A3B with the vLLM pooled setup with the four metric comparison (Token-DiFR, cross entropy, Token-DiFR likelihood, and exact token match).



(a) Standard misconfigurations.



(b) Bug simulations.



(c) Activation comparisons (AUC).

Figure 15. Token-DiFR and Activation-DiFR results for Qwen3-30B-A3B with the vLLM + HuggingFace pooled setup with the four metric comparison (Token-DiFR, cross entropy, Token-DiFR likelihood, and exact token match).