Logprob Majority-Consensus Verification (LMCV): A Low-Cost Execution-Verification Protocol for Open Inference Networks

TrueOpen Research

Decentralized Inference & Verification Group

Abstract

In decentralized compute networks and open inference markets, tasks are dispatched to untrusted worker nodes, raising a core question for service integrity: how can the network confirm that a worker *actually executed the user-specified reference model and runtime configuration*, without re-running the entire generation? Full recomputation costs as much as generating the answer again, while comparing only the final text cannot distinguish same-family models, quantized variants, or a small model masquerading as a large one. We present **Logprob Majority-Consensus Verification (LMCV)**, an execution-verification protocol that combines a *token-level probability fingerprint* with *independent multi-node consensus*. The key observation is that once generation is complete, every output prefix is fixed; a verifier can therefore run a single parallel *teacher-forced prefill* over the frozen input_ids + output_ids path, reconstruct each position's predictive distribution under the causal mask, and check the logprob, rank, and top-*k* candidate set of each returned token-without decoding token by token. LMCV constrains dishonest nodes with four mechanisms: (i) a deliver-first, choose-later *VRF verifier selection*; (ii) an anti-copying *commit-reveal* schedule; (iii) a registration-time *multi-metric decision* over numeric deviation, candidate compliance, and batch statistics; and (iv) a *two-of-three majority consensus* among independent verifiers. For Mixture-of-Experts (MoE) models we add a *routed-experts joint check*, using the layer-0 routing fingerprint as a structural criterion where per-token logprob alone is too noisy to yield a clean decision boundary.

On Qwen3-32B and Qwen3-8B (dense) and Qwen3.6-35B-A3B (MoE), the same-model BF16 replay baseline is narrow and stable across GPUs (6000ws / H100 / 4090 / L4), while FP8, AWQ, and heterogeneous smaller models separate sharply on p99, rank-delta rate, top-*k* Jaccard, and union-JS. For MoE, per-token logprob is confounded by sparse heavy-tailed spikes, but the layer-0 routed-experts fingerprint forms a structurally stable, cross-GPU-reproducible gap between same- and cross-model cases (under a layer-0 random-5 aggregate, a BF16 verifier attains TPR = 1.000 with 0.000 false-pass at threshold 0.060). On compute time, Qwen3-8B / Qwen3-32B benchmarks report one verification replay at **0.85%-8.42%** of generation time for outputs of **1,024-8,192 tokens**. A separate Qwen2.5-7B experiment reports **three replay passes combined at 1.9%-14.8%**, with output fixed at **1,000 tokens**. These configuration-dependent timing results are not measurements of total online-service cost (§6).

Keywords: execution verification; logprob fingerprint; majority consensus; commit-reveal; VRF; Mixture-of-Experts; routed experts; vLLM; decentralized inference

1. Introduction

1.1. Background

An open inference network lets any staked node accept model-inference jobs: a user specifies a reference model (weights version plus runtime configuration), the network dispatches the job to a worker that produces the result and bills per token. This openness creates a fundamental trust problem-**neither the user nor the protocol can directly observe what a worker executed internally.** A rational but dishonest worker has strong economic incentives to fake the service more cheaply:

- substitute a **smaller same-family model** (e.g. serving 8B as 32B) or a more aggressive **quantization** (FP8, AWQ posing as BF16) to cut memory and compute;
- generate tokens with a **cheap model first**, then run one prefill of the reference model to attach "correct-looking" probability evidence;
- return cached or task-irrelevant content.

The two naive defenses both fail. *Full recomputation* (having another node regenerate the same output) costs essentially the same as performing the task again-verification and generation are of the same order, which is economically infeasible. *Plain text comparison* cannot distinguish model identity under sampling, paraphrase, or the naturally high output similarity of same-family models, and is entirely defeated by the "cheap-generate then attach-probability" forgery above.

It bears emphasis that the object of verification is **execution compliance** (whether the agreed model and configuration were used), not **content correctness**. Factual errors or hallucinations produced by the reference model itself do not constitute node cheating.

1.2. Core idea

LMCV rests on a structural fact about autoregressive language models: **after generation completes, the prefix at every output position is fixed, so each position's next-token predictive distribution is a deterministic (for a given numeric implementation) recomputable quantity.** Let a worker return the token sequence $y_1, y_2, \ldots, y_L$. At position t the reference model, given input x and the true prefix $y_1 \ldots y_{t-1}$, yields a candidate distribution in which y_t has a definite logprob and rank. The verifier merely feeds input_ids + output_ids into the reference model in a single *teacher-forced prefill*, reconstructs all positions' distributions in parallel under the causal mask, and reads off each token's probability and rank-**with no token-by-token decoding.**

This demotes "verification" from "replaying generation" to "one forward pass," reducing measured compute time in the evaluated long-output configurations (§6). On top of that, LMCV uses cryptographic commitments, random selection, and majority consensus to turn it into a collusion-resistant, settleable, auditable protocol (§3-§5).

1.3. Contributions

1. **Protocol.** A complete LMCV design-deliver-first/choose-later VRF verifier selection, an anti-copying commit-reveal schedule, a result-commitment formula binding task/round/identity, and two-of-three majority consensus with post-hoc audit and correction (§3, §4.4).
2. **Decision rule.** A multi-metric numeric decision built from mean absolute deviation, high quantiles, rank-change rate, top-k overlap, and distributional divergence, plus a *per-position candidate check* tightened by k_{gen} to defeat the "cheap-generate then attach-probability" forgery (§4.1-§4.3).
3. **MoE routing joint verification.** An analysis of why per-token logprob alone cannot yield a clean MoE decision (sparse heavy tails, position-correlated spikes), and a structural criterion based on routed experts-in particular the **layer-0 routing fingerprint**-with an explanation of why layer 0 has a higher signal-to-noise ratio than all-layer aggregation (§5).
4. **Cost and empirics.** Separate compute-time results for one Qwen3 verification replay and three Qwen2.5-7B replay passes, with their model, output-length and measurement boundaries stated explicitly (§6), plus dense/MoE, cross-GPU evidence for fingerprint separability and stability (§7).
5. **Security analysis.** Resistance to model substitution, quantization substitution, probability forgery, verifier copying, and collusion-and the boundaries of those guarantees (§8).

2. Threat Model and Assumptions

2.1. Parties

- **User / SDK.** Submits the task, specifies the reference model and runtime configuration, escrows fees, receives and checks the result.
- **Worker.** Accepts the task, runs the model, produces the result, and retains computation evidence. May be dishonest.
- **Verifier.** Three independent nodes randomly selected by the protocol; each recomputes by prefill of the reference model and renders a decision. The worker of the task may not verify its own task.
- **Builder / Nexus.** Coordinates delivery and forwarding (signature, ordering, content-commitment checks); does not alter the verification outcome.
- **Chain & DA.** The chain fixes commitments, signatures, and decisions; data availability (DA) stores the full result and computation evidence within a retention window, with reads gated by phase and permission.

2.2. Adversary capabilities

The adversary controls some worker/verifier nodes and aims to bill for work performed below the cost of honest execution, or, as a verifier, to change the decision by copying or colluding. We assume the adversary can: choose any local model, quantization, inference engine, and hardware; read the inputs and results of tasks it accepts; and control a bounded fraction of the network's lottery weight (stake × performance score). We assume it **cannot** break the cryptographic assumptions of the hash/signature/VRF primitives, and **cannot** manipulate the randomness beacon of a designated future block.

2.3. Security goals

- **Soundness.** Non-compliant execution is rejected with high probability, or overturned by a post-hoc challenge.
- **Completeness.** Honest execution (including normal hardware/engine numeric jitter) passes with high probability.
- **Collusion resistance.** A worker cannot pre-arrange its checkers; a verifier cannot change its answer after seeing others' numbers.
- **Cost feasibility.** The marginal compute cost of verification is far below that of generation.

2.4. Premise

LMCV is a *statistical* execution verification, not a cryptographic proof of computation (the latter is handled by TrueOpen's separate “cryptographic-proof layered random verification” track). Its detection power comes from the check items and calibration data; majority consensus constrains the independence of node judgments, not the coverage of the checks themselves.

3. The LMCV Protocol

3.1. Overview

```
① Worker generates the result and delivers it (no need to wait for verification)
→ records worker evidence (logprob / rank / top-k [/ routed_experts])
② Worker writes the result digest and evidence root into a signed receipt → on-chain;
full material handed to DA
③ Protocol selects 3 official Verifiers via VRF (worker submits first, selection after)
④ Each Verifier recomputes by prefill independently → submits a "numeric commitment"
⑤ After the commitment set is fixed, the reveal phase begins:
worker reveals comparison numbers; verifiers reveal numbers + salt
⑥ Check commitments → apply registered numeric / candidate / batch rules → 2-of-3 consensus
⑦ Afterwards, any party with read permission may audit and challenge for correction
```

The key design points are: *delivery decoupled from verification* (the user gets the result promptly); *evidence committed first* (once on-chain, the material cannot be altered); *selection after submission* (the worker cannot arrange its checkers); and *numbers revealed last* (verifiers cannot copy).

3.2. Worker evidence

The worker decodes normally and, for each output position *d*, records:

```
input_ids, output_ids
worker_selected_logprob[d] # logprob of the actually returned token
worker_selected_rank[d] # its rank in the reference model's distribution
worker_topk_logprobs[d] # top-k candidates and their logprobs
(MoE, additionally) routed_experts[d][layer][slot] # expert ids and slot order
```

In experiments the generation side uses a deterministic configuration (temperature=0, top_p=1, top_k=0) for alignment; here top_k=0 follows the benchmark engine's convention for an unrestricted top-*k*, not the production candidate bound used by §4.2. In production the generation parameters are fixed at model registration and bound as part of the commitment. One **must use the original token sequence from generation**: different token compositions may map to the same text yet carry different probabilities and ranks, so a re-tokenized sequence is not a substitute; the verifier must also confirm the text reconstructed from that sequence matches what the user received.

3.3. Random verifier selection (deliver first, choose later)

If a worker could know in advance who will check, it might collude to bypass the rules. The protocol therefore **fixes the worker's result and evidence commitment first, then selects the checkers**, in two random draws:

1. **Candidate screening.** A verifiable random function (VRF) draws a candidate pool from nodes meeting stake, performance-score, liveness, model/config support, and penalty-state requirements.
2. **Official selection.** Candidates willing to verify raise their hands with a signature; once registration closes and the roster and weights are fixed, the randomness beacon of a *designated future block* selects 3 official verifiers by weight *without replacement*.

Participants can re-derive the draw from the fixed roster, weights, and public randomness. Collusion resistance depends on the malicious fraction of lottery weight and the manipulation resistance of the randomness source-**not on the total node count.**

3.4. Independent recomputation by prefill

A verifier feeds input_ids + output_ids into the user-specified reference model (weights version and runtime configuration) in a *single* prefill, obtaining each position's logprob, rank, and top-*k* candidate distribution. For output position *d*, the aligned position is:

```
prompt_pos = len(input_ids) + d
```

The causal mask guarantees that position t can see only x and $y_1 \ldots y_{t-1}$-so one parallel prefill still respects the generation-time ordering. What is compared is **the model distribution that worker and verifier assign along the same token path**, not the text similarity of two free generations.

The comparison covers **all generated tokens, without sampling**: every position must have a corresponding metric, missing ones included, so one cannot shrink the scope by omitting hard-to-pass tokens. (Cross-task batch statistics are a separate layer; see §4.3.)

3.5. Commit-reveal

Obtaining reference-model numbers is not enough-a verifier must be prevented from directly copying the worker's answer. Hence, before a verifier submits its commitment, **the worker's comparison numbers are not disclosed to it.** Each verifier first assembles its metrics and evidence into a result_payload_hash, then hashes it on-chain together with task information and a locally kept secret salt:

```
commit_hash = SHA256(
frame("SINGA_RESULT_COMMITMENT_V2")
|| frame(chain_id) || frame(task_id) || frame(task_hash)
|| frame(u32be(verify_round)) || frame(verifier_operator_address_bytes)
|| frame(result_payload_hash) || frame(salt)
)
```

- chain_id / task_id / task_hash: bind the chain, task id, and task content;
- verify_round / verifier_operator_address: bind the verification round and verifier identity;
- result_payload_hash: the digest of metrics and evidence, also binding the inference receipt, model configuration, and generation parameters;
- salt: a 32-byte random salt generated independently each time and kept secret until reveal, preventing pre-reveal enumeration;
- frame(·): prepends an 8-byte big-endian length to each field to remove concatenation ambiguity.

The selection randomness and the salt serve different purposes: the future-block VRF beacon drives publicly re-derivable selection and does not enter the commitment formula; the salt is generated locally by the verifier, kept secret until reveal, and cannot be derived from public information.

The commitment phase ends when all official verifiers' commitments are accepted, or the deadline passes. Fewer than two valid commitments fails the round; once the threshold is met the reveal phase begins and no new commitments are accepted for that round. Two existing commitments do not leak the answer to a not-yet-submitted third. At reveal, the chain recomputes the hash and rejects any mismatch. Thus **by the time each party sees the other's numbers, its own answer is already irrevocable.**

Under encryption, the token sequence needed for independent computation and the worker's comparison numbers are encrypted separately: the verifier first obtains the input, result, and original token sequence, computes independently, and commits; only after the reveal phase begins does the worker deliver the independent key for the comparison numbers-storing ciphertext early is not disclosing numbers early.

4. Verification Decision

After numbers are revealed and checked against commitments, the decision follows rules fixed at model registration. Metrics, tolerances, and batch conditions are all determined before the task starts, so a verifier cannot cherry-pick a favorable method after the fact.

4.1. Numeric comparison metrics

Let the worker/verifier logprob at output position t be ℓ^W_t and ℓ^V_t, over N valid comparison positions.

Mean absolute deviation (overall departure):

$D_{mean} = (1/N) \sum_{t=1}^{N} |\ell^{W}_{t} - \ell^{V}_{t}|$

A mean can hide a few large deviations, so the following metrics are added (all from the same prefill):

Metric	Meaning
p95 / p99 / p999	High quantiles of abs_logprob_diff, characterizing tail departure
rank_delta_rate	Fraction of positions where the selected token's rank changed
topk_jaccard	Jaccard overlap of worker/verifier top-*k* candidate sets
union_js	Jensen-Shannon divergence over the union of top-*k*
missing_selected_count	Number of positions where the verifier did not return the worker's selected token

Note that we compare vLLM-exposed selected/top-*k* logprobs, not full-vocab raw logits.

4.2. Per-position candidate check (defeating probability forgery)

Matching numbers do not prove the tokens were produced by the specified model. An attacker can **generate tokens with a cheap model first, then prefill the reference model to fetch the true logprobs of those tokens as evidence**-defeating the D_{mean} check. Detecting this relies on the candidate check: a meaningful fraction of a small model's tokens fall outside the reference model's top-*k*. Let the registered candidate count be k_{gen}; then at every position we require:

$\text{rank}_{M}(y_t \mid x, y_1, \ldots, y_{t-1}) \le k_{gen}$

To make this check strongly binding, the network **tightens the sampling top-*k* used at generation into a registered execution requirement, setting k_{gen} equal to that bound.** Every token of a compliant output then lies within the reference model's top-*k*, whereas deviating tokens from a substitute model are stably detected by the per-position candidate check and batch statistics. If a model is registered for greedy decoding, the candidate rule can be expressed as k_{gen}=1, or as another explicitly registered finite bound with its own calibrated tolerance; an unrestricted experimental top_k=0 is not by itself a strong production candidate rule. Positions clearly out of range constitute grounds for rejection; positions near the boundary are handled by registered tolerance and do not pass automatically just because other positions are normal.

4.3. Batch statistics

A single task is subject to chance jitter. Multiple tasks from the same worker under the same model and verification configuration form a *statistical batch* per registered rules, with batch membership fixed before verification. The model maintainer sets the batch size, minimum valid sample count, and accept/reject conditions by testing. Batch statistics judge whether the overall deviation lies in the normal range, while the per-token check retains constraints on specific violating positions-**the two cannot substitute for each other.** Missing data, computation failures, and unavailable material are handled separately; one cannot reach a “pass” by deleting failed samples.

The decision is ternary: all pass conditions met → accept; reject conditions met → reject; insufficient evidence or in-between → pending, with supplementary checks per rule.

4.4. Majority consensus and post-hoc audit

After the statistical decision, the protocol reconciles the valid results of the three official verifiers; **at least two forming a rule-consistent identical result** confirms majority consensus and the corresponding billable work. Batch statistics handle numeric jitter; multiple independent verifiers constrain a single node's judgment-the two are complementary.

Majority consensus is not final: the chain fixes each party's commitments, signatures, and decisions, and DA retains the material within a window; a party with read permission can redo the prefill with the reference model and audit, and upon finding a challenge-eligible error, submit evidence to request settlement

correction and accountability. A majority agreement therefore does not strip an erroneous conclusion of the possibility of discovery and correction.

5. MoE Routing Joint Verification

5.1. Why per-token logprob is insufficient for MoE

Dense models pass every token through the same dense layers; numeric differences arise mainly from GPU/kernel/dtype/quantization error, so the same-model BF16 replay baseline is narrow and a smaller or re-quantized model produces a stable distributional shift over many tokens. MoE models additionally route each token through a subset of experts: even with identical weights, differences in decode-time vs. prefill-time routing, batch composition, and numeric path can push a few tokens into different expert sets, producing **local logprob spikes** (see §7.2). This makes the MoE same-model baseline inherently wider and heavier-tailed, so single-token or simple-quantile thresholds waver between false reject and false accept.

logprob is the final projection of the output distribution and cannot directly explain whether a spike arises from model difference, path difference, or a routing-sensitive point. Routed experts, by contrast, expose MoE's structural intermediate decision directly:

```
input/context → router → selected experts
```

Differences in model, quantization, or run path appear in expert selection *before* the final logprob, so routing fingerprints are more natural for MoE identity.

5.2. Method

The worker records expert ids and slot order at generation and commits them; the verifier obtains an independent routing record from the reference-model prefill. After reveal, the two records are aligned by global token position, comparing commonly covered positions and preserving slot order (so that different record start/end points are not mistaken for expert-selection errors). Positions that are missing, uncovered, or impossible to align are not silently dropped; they are accounted for as insufficient evidence or as rejection conditions under the registered rule. Expert selection is discrete, and normal numeric deviation can flip individual slots; the method therefore aggregates the inconsistent-slot ratio over a batch:

$$R = (\sum_i m_i) / (\sum_i e_i)$$

where m_i is the number of slots in sample i where worker and verifier chose different experts, and e_i is the number of slots actually compared in that sample. Aggregating by actually compared slots avoids weighting short and long samples equally. Which layers to compare, how many samples, and how large a routing difference to allow are all determined by model testing. A batch is accepted only if it **simultaneously** satisfies the routing-difference threshold and the output-candidate rules of §4, with sufficient evidence and no pending positions.

5.3. Why layer 0 beats all-layer aggregation

The experiments in §7.3 show the layer-0 routing fingerprint is cleaner than all-layer aggregation, for three reasons:

1. **Error accumulation.** A later layer's routing input is already affected by earlier layers' outputs; even with an identical model, decode/prefill, tiny hardware differences, and batch composition can push some tokens to the other side of a router decision boundary in later layers, and these mismatches raise the same-model baseline.
2. **Dilution.** Averaging over all layers washes out the most discriminative early-layer differences. Layer 0 acts directly on the embedding and raw context representation, before any error propagation from multi-layer expert selection-more like a stable “initial routing signature.”

3. **Over-strict metric.** Cross-layer per-token exact match rapidly approaches a low floor as layers increase, collapsing dynamic range and reducing discrimination; layer-0 entry-mismatch and set-match retain higher dynamic range and are better suited to thresholding.

Hence for MoE we **recommend the layer-0 route as the primary criterion**, with all-layer route drift as auxiliary diagnostics (to analyze the source of drift) and window-level logprob statistics as a secondary criterion.

6. Cost Analysis

The economic basis for LMCV's feasibility is that **one prefill is far cheaper than token-by-token generation.** Based on timings from benchmark_worker_verifier_latency.py on Qwen3-8B (BF16, single RTX 4090, vLLM offline V1 engine, top-*k*=20, batch=1):

- **Worker cost ≈ output_len × 17.5 ms:** time is spent almost entirely on *serial decode*, largely independent of input length (fixing out=1024, input 128→8192 raises worker only from 17.63s to 19.80s, +12%).
- **Verifier cost ≈ total_tokens / ~5200 + fixed overhead:** a single *parallel prefill* computes all input+output positions at once, at ~5200 tok/s.

The two differ by about two orders of magnitude, and the longer the output the better verification pays off:

Table 1. Representative worker-vs-verifier cost ratios (Qwen3-8B, RTX 4090, batch=1).

Scenario (input/output)	Worker gen (s)	Verifier (s)	Ratio
512 / 1024	17.71	0.217	**81.5×**
128 / 2048	35.40	0.312	**113.6×**
2048 / 4096	73.04	0.819	**89.2×**
1024 / 4096	72.24	0.691	**104.6×**

For output=1, the reported worker/verifier ratio is 0.62-0.82: verification can take longer than generation. The representative rows above do not define a universal verification ratio.

Across the published **Qwen3-8B / Qwen3-32B** timing tables, restricted to outputs of **1,024-8,192 tokens**, one replay takes approximately **0.85%-8.42%** of generation time. The models were measured on an RTX 4090 and an H100 NVL, respectively. See the benchmark report and whitepaper §3.1.5.

A **separate Qwen2.5-7B** experiment reports **1.9%-14.8%** for **three replay passes combined**, calculated as 3 × tf_full / gen, with output fixed at **1,000 tokens** and input lengths of 50-8,000 tokens. It used FP16, batch size 1, vLLM 0.20.1 and one RTX PRO 6000 Blackwell (96 GB). See the 7B experiment report. This range is not obtained by multiplying the Qwen3 ratios by three.

Measurement scope. Both results concern compute time under their respective configurations. Neither includes networking, gas, queuing, evidence handling or settlement costs. Three replay passes do not establish an end-to-end service running on three independent nodes. Continuous batching and production scheduling may change the ratios; production total cost and throughput must be measured separately.

7. Empirical Evaluation

7.1. Dense models: the logprob fingerprint works

Qwen3-32B (worker = BF16 @ 6000ws, 300 prompts, input buckets 32-16384, logprobs=64). Using 6000ws→6000ws BF16 as the baseline:

Table 2. Qwen3-32B verifier metrics. Same-model BF16 forms a narrow, cross-GPU-stable envelope; quantized and smaller models separate sharply.

Verifier	Quant	mean	p99	p999	rankΔ	jaccard p05	union_js p99
Qwen3-32B (BF16, 6000ws)	BF16	0.0102	0.1167	0.2478	0.0081	0.8824	0.0035
Qwen3-32B (BF16, H100)	BF16	0.0099	0.1140	0.2243	0.0070	0.8824	0.0033
Qwen3-32B-FP8	FP8	0.0268	0.2997	0.7665	0.0205	0.7067	0.0217
Qwen3-32B-AWQ	AWQ	0.0671	0.7962	2.2053	0.0478	0.5238	0.0921
Qwen3-14B	BF16	0.2608	3.9261	8.1296	0.1165	0.3061	0.4250
Qwen3-8B	BF16	0.3749	5.8105	11.6767	0.1402	0.2549	0.5189

p99 multiples relative to baseline: FP8 ≈ 2.57×, AWQ ≈ 6.82×, 14B ≈ 33.6×, 8B ≈ 49.8×. Same-model BF16 forms a narrow, consistent legitimate drift envelope on both 6000ws and H100; FP8/AWQ/smaller models separate sharply.

Qwen3-8B (worker = BF16 @ 4090, verifiers on 4090/L4) confirms the same conclusion: the same-model BF16 baseline is narrow with tiny cross-GPU differences (4090 vs. L4 mean abs diff 0.0081 vs. 0.0090), while FP8/AWQ/Qwen2(.5)-7B separate progressively (Qwen2.5-7B mean p99 reaches 5.83). One can therefore set explicit batch-level pass/reject thresholds, e.g. pass requires mean_abs_diff ≤ 0.015, p99 ≤ 0.200, rank_delta ≤ 0.012, topk_jaccard ≥ 0.940.

7.2. MoE: signal present, but hard to threshold

Qwen3.6-35B-A3B, 300 samples (4 BF16 replicates + FP8):

Table 3. MoE 300-sample logprob metrics. FP8 shifts right, but the same-model BF16 baseline is already heavy-tailed.

Run	Quant	mean	p50	p99	p999	max	rankΔ	union_js p99
BF16 ×4 (range)	BF16	0.054-0.062	~5e-4	0.37-0.43	10.2-11.9	24-26	0.022-0.025	0.053-0.064
FP8	FP8	0.0848	0.0011	0.6729	12.68	27.56	0.0364	0.1165

FP8 does shift right relative to BF16, but the **same-model BF16 baseline is itself heavy-tailed** (p50≈5e-4 yet p999≈10-12, max≈24-26). Moreover, among abs_diff>1 spike locations, 65.1% appear in at least 2 of the 4 BF16 replicates and 21.5% in all four-indicating spikes correlate with specific token/context (most likely routing boundaries / decode-prefill path / kernel scheduling) rather than pure random noise. A window-median suppresses the BF16 tail and reveals FP8's systematic shift (e.g. at window=16 BF16 p99 drops to ~0.05 while FP8 is ~0.08), but this remains a statistical separation, not the clean boundary seen in dense models. The 5000-sample moe_1 experiment (BF16 worker → BF16/FP8 verifier; FP8 worker → BF16 verifier) reinforces the same conclusion. **MoE should therefore not rest its final decision on logprob alone.**

7.3. MoE routing fingerprint: a structural criterion

The routed-experts reports cover BF16/FP8 same-model and cross-model comparisons on 6000ws / H100 (1000 samples per case, ~124k tokens). All-layer entry-mismatch:

- BF16 same-model ≈ 0.189-0.192
- FP8 same-model ≈ 0.266-0.273
- BF16/FP8 cross-model ≈ 0.292-0.294

All-layer already separates BF16-same from cross, but FP8-same vs. cross differ by only ~0.02. **The layer-0 fingerprint widens the gap:**

Table 4. Layer-0 routed-experts metrics. Same-model FP8 vs. cross-model separate far more clearly at layer 0.

Case	Layers	entry_mismatch	set_match	token_match	topk_jaccard
6000 bf16 vs fp8 (cross)	0	0.1134	0.8871	0.5912	0.9744
6000 fp8 vs fp8 (same)	0	0.0492	0.9504	0.8009	0.9890

At layer 0, FP8 same-model entry-mismatch = 0.0492 vs. cross = 0.1134, an absolute gap widening to 0.0642 (cross/same ≈ 2.30×); the BF16 verifier's layer-0 positive mean is ~0.024-0.029 vs. FP8 negatives ~0.115-0.117, an even clearer separation. In a **cross-GPU random-5 threshold test**-using only layer 0, aggregating entry-mismatch over 5 randomly sampled examples-the BF16 verifier attains combined TPR = 1.000000 with false-pass = 0.000000 at threshold 0.060; the FP8 verifier attains TPR = 0.999453 with false-pass = 0.000000 at threshold 0.080. The layer-0 routing fingerprint is thus stable and usable across the 6000ws/H100 comparison.

7.4. Summary

Table 5. Recommended verification design by architecture.

Architecture	Primary criterion	Auxiliary
Dense	Multi-metric logprob envelope (mean/p95/p99/rankΔ/jaccard/union_js) + candidate check	-
MoE	**Layer-0 routed-experts fingerprint** + candidate check	All-layer route drift (diagnostic); window-level logprob statistics

8. Security Analysis

- **Model substitution (large→small, heterogeneous).** Produces a stable distributional shift over many tokens; in the dense case p99/rankΔ/union_js separate from the baseline by tens of times (§7.1), stably detected by batch statistics; a small model's tokens also fall out of the reference top-k in bulk, caught per-position by the candidate check (§4.2).
- **Quantization substitution (FP8/AWQ posing as BF16).** In dense, FP8 ≈ 2.6× and AWQ ≈ 6.8× p99 separation allow a hard decision; in MoE, decided by the layer-0 routing fingerprint (cross/same ≈ 2.3×) and the batch routing-difference rate R.
- **Probability forgery (cheap generation + reference-model attach).** D_{mean} is fooled, but the per-position candidate check tightened by k_{gen} exposes the substitute model's out-of-range tokens-the key design of §4.2.
- **Verifier copying the worker's answer.** Commit-reveal keeps the worker's comparison numbers undisclosed until after verifier commitment, the salt prevents pre-reveal enumeration, and the chain recomputes and checks the hash at reveal (§3.5).
- **Worker pre-selecting checkers / collusion.** Deliver-first/choose-later, two-stage VRF, a future-block randomness beacon, and without-replacement selection (§3.3); collusion resistance depends on the malicious lottery-weight fraction and randomness-source manipulation resistance, not the total node count.
- **Single-point misjudgment / chance jitter.** Two-of-three majority consensus + batch statistics + ternary decision (accept/reject/pending) + post-hoc challengeable audit (§4.4).

Boundary. LMCV is statistical verification, not a proof of computation; its detection power is bounded by the check items and calibration data, and majority consensus does not extend the coverage of the checks. Cross-GPU/engine tolerances must be calibrated with ample positive/negative samples, or honest execution may be falsely rejected and borderline forgeries let through.

9. Discussion and Related Work

LMCV sits between replay-based verification and cryptographic proof: versus full recomputation it reduces measured compute time in the evaluated long-output configurations (§6); versus zkML / verifiable computation it offers no cryptographic soundness but adds almost no proving overhead, making it suitable as a *first-layer* verification for cost- and throughput-sensitive open inference networks. The two are complementary within TrueOpen: fixed-point-compliant quantized models can take the cryptographic-proof layered random verification track, while floating-point / reference-model paths take LMCV.

Recent work on verifiable LLM inference is moving close to LMCV's problem setting. VeriLLM [1] also targets decentralized inference, keeps verification cost near the 1% regime, and exploits the structural gap between prefill and autoregressive decode. LMCV shares the idea of replacing full generation with a cheaper replay-like check, but makes the worker's selected/top-*k* logprobs, ranks, and MoE routed_experts a registered model-identity fingerprint that can be settled by majority consensus. DiFR [2] studies inference verification despite numerical nondeterminism and introduces Token-DiFR and Activation-DiFR; like LMCV, it treats normal numeric drift as an unavoidable part of verification and uses token/activation fingerprints for audit, while LMCV focuses on open-inference settlement and embeds commit-reveal, random verifier selection, and two-of-three consensus at the protocol layer.

Pal et al. [3] argue that privacy-preserving inference mechanisms can enable cheap verifiable LLM inference with only a small amount of additional token work. Their threat model, in which a service provider may replace a requested large model with a cheaper smaller one, is directly aligned with LMCV's candidate-check motivation. CommitLLM [4] follows a related commit-and-audit direction: it binds checkpoint, quantization, runtime configuration, input processing, sampling randomness, decode policy, and delivered text into a compact receipt, then opens selected execution trace material upon challenge. LMCV and CommitLLM both rely on committing before audit/reveal to prevent post-hoc rewriting; CommitLLM is closer to per-response trace-bound auditing, whereas LMCV uses independent reference-model prefill by multiple verifiers and settles through majority consensus.

Cryptographic-proof and TEE-based systems provide stronger but heavier guarantees. NanoZK [5] and OpenLLM [6] advance verifiable LLM inference through layerwise zero-knowledge proofs and modular zkSNARKs, respectively; these approaches can offer a security model closer to cryptographic soundness, but proof cost, quantization and approximation constraints, operator coverage, and production-inference-stack integration remain major engineering barriers. VeriAttn [7] instead uses TEE-GPU cooperation to verify Transformer attention. Compared with TEE approaches, LMCV does not rely on a specific hardware root of trust but on independent multi-node recomputation and random selection, avoiding a single supply-chain trust assumption-at the cost of statistical rather than cryptographic or deterministic guarantees.

A separate line of work shows why LMCV must bind and calibrate the execution environment itself. Yuan et al. [8] analyze numerical nondeterminism caused by GPU count, GPU model, batch size, and finite-precision floating-point computation. Pape et al. [9] call the inference backend a “silent hyperparameter” and show that backends can change token probabilities and outputs even when weights, decoding parameters, and hardware are held fixed. Wimbauer et al. [10] and Zhang et al. [11] further show that inference engines, attention backends, GPU models, and software stacks leave observable fingerprints. These results support LMCV's requirement to register weights, tokenizer, chat template, precision, engine, prefix-cache behavior, batching policy, and routing return format; they also explain why thresholds must be calibrated on positive and negative samples across GPUs and engines rather than set from theory alone.

10. Limitations

1. The logprob experiments use vLLM-exposed selected/top-*k* logprobs, not full-vocab raw logits.
2. The main experiments cap output at ~128 tokens; long-output scenarios need separate calibration.
3. The dense 32B main experiment centers on 6000ws worker evidence and does not cover all worker/verifier directions.
4. MoE logprob already fails to form a clean hard boundary on the same GPU; its cross-GPU stability should rest mainly on routed_experts (especially layer 0).
5. Productionizing routed_experts requires fixing/calibrating the vLLM version, prefix cache, batching policy, and routing return format.
6. This is statistical identity verification, not cryptographic proof; collusion resistance depends on an unmanipulable randomness beacon and a bounded malicious lottery weight.

11. Conclusion

LMCV recasts “verifying that a worker used the specified reference model” as a single parallel prefill recomputation over a frozen token path, and-via VRF random selection, commit-reveal, registration-time multi-metric decision, and two-of-three majority consensus-builds it into a collusion-resistant, settleable, auditable protocol. Empirically, the dense logprob fingerprint is narrow and cross-GPU-stable with sharp separation of quantized and smaller models, allowing a direct hard decision; MoE cannot be hard-decided by logprob alone due to sparse heavy tails, but the layer-0 routed-experts fingerprint provides a cross-GPU-stable structural criterion. On compute time, the Qwen3-8B / Qwen3-32B results show one replay at 0.85%-8.42% of generation time for outputs of 1,024-8,192 tokens. The separate Qwen2.5-7B experiment shows three replay passes combined at 1.9%-14.8% with output fixed at 1,000 tokens. These measurements support further evaluation of verification overhead; they do not establish production total cost or the economics of a deployed three-node service.

Worker-verifier verification should therefore be designed by architecture: **dense relies primarily on the logprob fingerprint; MoE relies primarily on the layer-0 routed-experts fingerprint, aided by all-layer route drift and window-level logprob statistics**-all settled and corrected within a unified commit-reveal + majority consensus framework.

Appendix A Notation

Symbol	Meaning
x	User input token sequence
$y_1 \cdots y_L$	Output token sequence returned by the worker
ℓ^W_t, ℓ^V_t	Selected logprob of worker / verifier at position t
D_{mean}	Mean absolute logprob deviation
k_{gen}	Registered candidate count (= sampling top-k bound)
$\text{rank}_M(\cdot)$	Rank of a token under reference model M
R	Batch expert-routing inconsistency rate
m_i, e_i	Inconsistent slots / compared slots in sample i

Appendix B Primary Data Entry Points

Data	Path
Dense 32B worker-verify	worker_logits_verify/results/qwen3_32b_identity_v1/worker_verify/
Qwen3-8B worker-verify	qwen3_8b_4090_data_results/results/qwen3_8b_identity_v1/worker_verify/
MoE 300-sample worker-verify	results/qwen3.6-35b-a3b-bf16-6000ws/worker_verify/
MoE 5000-sample data / results	moe/moe_1_data.zip / moe/moe_1_results.zip
Routed-experts reports	moe/reports/
Latency benchmark script	vllm_worker_verify_pipeline/benchmark_worker_verifier_latency.py

References

1. Ke Wang, Zishuo Zhao, Xinyuan Song, Zelin Li, Libin Xia, Chris Tong, Bill Shi, Wenjie Qu, Eric Yang, Lynn Ai. VeriLLM: A Lightweight Framework for Publicly Verifiable Decentralized Inference. arXiv:2509.24257, 2025.
2. Adam Karvonen, Daniel Reuter, Roy Rinberg, Luke Marks, Adrià Garriga-Alonso, Keri Warr. DiFR: Inference Verification Despite Nondeterminism. arXiv:2511.20621, 2025.
3. Arka Pal, Louai Zahran, William Gvozdjak, Akilesh Potti, Micah Goldblum. Privacy-Preserving Mechanisms Enable Cheap Verifiable Inference of LLMs. arXiv:2602.17223, 2026.
4. CommitLLM. CommitLLM: Verifiable execution for LLM inference. Technical report / system prototype, 2026.
5. Zhaohui Wang. NanoZK: Privacy-Preserving Verifiable Inference for Large Language Models via Layerwise Zero-Knowledge Proofs. arXiv:2603.18046, 2026.
6. Yunbo Yang, Yupeng Ren, Changtong Xu, Rui Zhang, Xuanming Liu, Jin Tan, Tao Wei, Bingsheng Zhang, Kui Ren. OpenLLM: Modular and Scalable zkSNARKs for Verifiable LLM Inference. IACR Cryptology ePrint Archive, 2026/1578, 2026.
7. Ziqun Chen, Ming Wu, Michael Heinrich, Jason Zeng, Huiying Lan, Tianwei Zhang, Rui Tan. Communication-Efficient Verifiable Attention for LLM Inference. arXiv:2606.16352, 2026.
8. Jiayi Yuan, Hao Li, Xinheng Ding, Wenya Xie, Yu-Jhe Li, Wentian Zhao, Kun Wan, Jing Shi, Xia Hu, Zirui Liu. Understanding and Mitigating Numerical Sources of Nondeterminism in LLM Inference. NeurIPS 2025.
9. David Pape, Jonathan Evertz, Lea Schönherr. The Silent Hyperparameter: Quantifying the Impact of Inference Backends on LLM Reproducibility. arXiv:2605.19537, 2026.
10. Anna Wimbauer, Jonas Möller, Erik Imgrund, Konrad Rieck. Fingerprinting Inference Systems of Large Language Models. arXiv:2605.29979, 2026.
11. Cheng Zhang, Hanna Foerster, Robert D. Mullins, Yiren Zhao, Ilia Shumailov. Hardware and Software Platform Inference. ICML 2025, PMLR 267:74882-74895.
12. TrueOpen Research. *A Worker-Verifier Model-Identity Study Based on Logprob and Routed Experts.* Companion technical report, 2026.
13. TrueOpen Research. *Worker vs. Verifier Latency Cost Benchmark.* Companion benchmark report, 2026.

14. vLLM Project. *vLLM: Easy, Fast, and Cheap LLM Serving with PagedAttention.* 2023-2026. (V1 engine; prompt_logprobs, --enable-return-routed-experts.)

15. Qwen Team. *Qwen3 / Qwen3.6 Technical Reports.* 2025-2026.