CommitLLM: A Commit-and-Audit Protocol for Open-Weight LLM Inference

Federico Carrone, Diego Kingston, Manuel Puebla, Mauro Toscano

LambdaClass
Universidad de Buenos Aires, Centro de Criptografía y Sistemas Distribuidos

Preprint

Abstract — Large language models are increasingly used in settings where integrity matters, but users still lack technical assurance that a provider actually ran the claimed model, decode policy, and output behavior. Fingerprinting and statistical heuristics can provide signals, but not exact per-response verification; zero-knowledge proof systems provide stronger guarantees, but at prover costs that remain impractical for production LLM serving.

We present CommitLLM, a cryptographic commit-and-audit protocol for open-weight LLM inference. CommitLLM leaves the provider on the normal serving path and verifies on the client's CPU. Instead of generating a proof per response, it commits to the execution trace and audits a sampled fraction of responses, checking large matrix products with Freivalds-style algebraic fingerprints. For Llama 70B a challenged token costs about 1.3 ms under the routine 10-layer audit, and about 10 ms for a 1-token full audit. The main costs are retained-state memory over the audit window and audit bandwidth, not per-response proving. Online tracing adds roughly 12–14% during generation in the current prototype; the larger commitment and finalization cost is measured separately, currently runs synchronously, and is a candidate for asynchronous deferral in production. The protocol is commitment-bound end-to-end. Within that binding, large linear layers are verified by verifier-secret, information-theoretically sound algebraic checks; quantization/dequantization boundaries and supported nonlinear subcomputations are checked by canonical re-execution; sampled decode is verified exactly via captured GPU logits plus an algebraic LM-head binding; and arbitrary-position attention outputs on stock GPU kernels are explicitly **not** verified. Stock-mode attention is instead audited from the inputs and wiring side: score anchoring against witnessed pre-softmax scores, KV provenance against committed cache rows, and GQA / RoPE-config / causal-mask wiring checks. Under the current witness contract the score-anchor and causal-mask audits are scoped to the last generated token. Routine prefix-state provenance is statistical unless deep audit is used. Unsupported semantics fail closed.

1 Introduction

Open-weight LLM inference presents a trust problem: the client sends a prompt to a provider who claims to run a specific model (e.g., Llama 70B [1]), but the client has no way to verify that the provider actually used those weights. The economic incentive to cheat is clear: serving a smaller or more aggressively quantized model reduces compute costs while the client pays the same price.

Existing approaches to verifiable inference fall into two categories. Cryptographic proof systems and specialized verifiable-inference systems such as SafetyNets [2] and zkCNN [3] can prove arbitrary or restricted computations, but impose overhead that makes real-time LLM inference impractical. Trusted execution environments (TEEs) provide attestation [4] but require hardware trust assumptions and limit deployment flexibility.

CommitLLM takes a different approach: a cryptographically bound sidecar commit-and-audit protocol that runs alongside unmodified GPU inference. The provider does not rewrite the model into a proving circuit, change the serving kernels, or generate a proof for every response; the normal serving path stays intact and the extra work is trace capture plus rare audit openings. CommitLLM borrows the cryptographic verification mindset of proof systems, but replaces full proof generation with commitment binding, direct audit, and cheap algebraic checks. The key insight is that INT8 quantized inference consists of operations that are either deterministic or canonically recomputable, and can be verified exactly. The only non-deterministic component on stock GPU kernels is FP16/BF16 attention. After every attempted production verification path for the attention interior was closed (exact stock-kernel replay, tiled / LSE replay, stock-bounded certification, and deterministic kernels), the kept claim is narrower and honest: arbitrary-position attention outputs are not verified, and attention is instead constrained from the inputs and wiring side.

The protocol provides exact, audited-but-not-verified, statistical, and fail-closed guarantees in different parts of the pipeline. The honest decomposition is: exact verification everywhere outside the attention interior; on the attention interior, audited inputs/wiring with arbitrary-position outputs explicitly out of scope; and statistical prefix anchoring unless deep audit is used. This is stronger than ordinary operational spot-checking or replica agreement: the provider commits before learning the audit challenge, and opened regions must satisfy cryptographically bound algebraic checks.

2 System Model

2.1 Threat Model

The adversary controls the inference server. They may swap model weights, modify attention computation, fabricate intermediate activations, or lie about earlier context. The adversary may cheat adaptively, choosing different strategies for different responses. The following timing and knowledge constraints apply:

1. **Commitment precedes challenge.** The provider sends the receipt (R_T, R_{KV}, M, $H(s)$, N, ...) before learning whether the response will be audited.

2. **Unpredictable selection.** The provider cannot predict which response, token position, or layer the verifier will challenge.
3. **Secret key.** The verifier holds r_j and $v_j^{(i)}$ secret; the provider never observes them or the audit decision rule.
4. **Local verification.** The verifier checks locally; no third party sees the challenge, the opening, or the conversation.

In the intended deployment, the client's verifier software automatically and randomly audits a small fraction of responses (e.g., 1–5%).

The client holds a verifier key ($\sim$ 48 MB for Llama 70B in the current benchmark projection) derived from the public model weights and secret verifier-side randomness. The client audits responses directly; no trusted third party sees the conversation. The attester / verifier / relying-party terminology here follows standard remote-attestation usage [4].

CommitLLM provides interactive, client-held verification. The client who holds the verifier key checks the computation directly. The receipt alone is not a succinct SNARK-like transferable proof that third parties can verify offline; this is a deliberate tradeoff for low overhead. However, the protocol is transferable in a weaker sense: if a full audit transcript is disclosed, third parties can independently re-check that disclosed audit against the public weights and committed receipt. What transfers is a bulky audit bundle rather than a compact proof object. Audit openings contain intermediate activations from which the conversation content can be reconstructed; this is why the client audits themselves rather than delegating to a third party by default.

The audit protocol is interactive, but the Freivalds randomness is not sampled publicly at audit time. It is precomputed into the verifier key and remains verifier-secret, so Freivalds acts here as a verifier-secret randomized algebraic check inside a cryptographically bound protocol rather than as a public non-interactive proof.

2.2 Design Principles

CommitLLM is designed around three constraints:

1. **No serving-path changes.** The provider runs unmodified GPU inference. The only addition is a tracing layer that captures intermediates alongside normal execution; there is no proving circuit, proof generation pass, or required kernel rewrite on the serving path.
2. **Modest online overhead.** Each response carries a compact receipt binding trace commitments, deployment specs, transcript randomness, and token count. The measured online cost of tracing and hooks is currently about 12–14% across the tested 7B/8B/70B runs; heavier receipt finalization occurs after generation.
3. **Client-side verification.** The client performs all verification using CPU-feasible operations (dot products, hash checks, and, for the audit-only attention bracket, score-anchor matrix arithmetic scaling as $O(n)$ at the audited token).

Protocol guarantees at a glance.

1. **Exact.** Preprocessing, model identity, shell matmuls, bridge operations, the final-token tail, sampled decode (via captured GPU logits plus LM-head Freivalds binding), and decode/output replay.
2. **Audited but not verified (stock-mode attention).** Arbitrary-position attention outputs are explicitly not verified. The attention interior is instead constrained by audits on attention inputs and wiring: score anchoring against witnessed pre-softmax scores, KV provenance against committed cache rows, and GQA / RoPE-config / causal-mask wiring checks. Under the current witness contract the score-anchor and causal-mask audits are scoped to the last generated token (Section 6).
3. **Statistical.** Prefix KV correctness under routine audit sampling (k prefix positions checked).
4. **Outside protocol scope.** Standard cryptographic assumptions, verifier-secret secrecy, no side-channel leakage of verifier secrets, and correct verifier execution.

2.3 Protocol Boundary

The final protocol uses four guarantee classes:

1. **Exact.** A property is cryptographically bound and then checked by information-theoretically sound algebraic verification or canonical recomputation with fully specified semantics.
2. **Audited but not verified.** The protocol does not verify the property's value; instead, it commits to and audits the **inputs** and **wiring** that surround it. For stock-mode attention this means score anchoring (recomputing $QK^T/\sqrt{d}$ from shell-verified Q and committed K against witnessed pre-softmax scores), KV provenance, and GQA / RoPE-config / causal-mask wiring checks; arbitrary-position attention outputs are explicitly out of scope.
3. **Statistical.** Commitment binding is exact, but correctness of unopened positions depends on challenge sampling unless deep audit is used. This is a coverage-probability distinction, not a floating-point approximation.
4. **Fail-closed.** A feature is either replayed exactly or rejected explicitly; the verifier never silently accepts unsupported semantics.

CommitLLM does *not* assume honest provider hardware or honest provider runtime behavior. Those are what the protocol is designed to remove. The explicit assumptions outside protocol scope are standard cryptographic assumptions, secrecy of verifier-only material, resistance to side-channel leakage of verifier secrets, and correct execution of the verifier itself.

The final protocol targets autoregressive decoder-only transformers with a committed capture layout and replay specification. Unsupported architectures must fail closed rather than be silently interpreted as if they shared decoder-only semantics.

Symbol	Meaning
R_W	Merkle root over model weights (public identity)
R_T	Merkle root over trace intermediates
R_{KV}	Merkle root over per-token K, V history
M	Deployment commitment over input/model/decode/output specs
$H(s)$	Transcript-seed commitment
N	Token count in response
c, ℓ	Number of challenged tokens; number of opened layers per token
m	Output dimension of the checked weight matrix (length of r_j)
k	Sampled prefix positions for KV provenance
p	Prime modulus for Freivalds checks ($p = 2^{32} - 5$ in the current implementation); arithmetic in $\mathbb{F}_p$

Table 1: Notation

3 Verification Layers

CommitLLM's verification is structured in six layers. This section defines the taxonomy and key terms; the full procedure is specified in Section 4.

The **shell** is the non-attention path through each transformer layer: seven INT8 weight matrix multiplications (W_q, W_k, W_v, W_o, W_{gate}, W_{up}, W_{down}), the exact INT8 bridge tensors that feed later INT8 stages (x_{attn} and x_{ffn}), the committed post-attention output attn_out_i8 that feeds W_o, and canonically recomputable operations such as Q/K/V dequantization, RoPE, RMSNorm, and SiLU. The shell includes nonlinear operations (RMSNorm, SiLU), but all exact shell operations are deterministic or canonically recomputable. Its exactness is a composition: information-theoretically sound algebraic checks (Freivalds [5]) on the weight matmuls, plus deterministic or canonical recomputation on the exact bridge operations.

Attention (QK^T, softmax, αV) is computed in FP16/BF16, which is not bit-reproducible across hardware. It is the only non-exact component, and arbitrary-position attention outputs are explicitly not verified in the kept protocol. The final exact tail starts from a captured pre-final-norm residual after the last layer, applies final RMSNorm exactly, binds the LM head with Freivalds, and replays the decode/output policy. For sampled decode, the exact token check is performed against captured GPU logits bound back to the LM head; for greedy decode, cheaper validated logits paths may be used where explicitly supported.

The six layers are:

1. **Input/model binding (exact).** Preprocessing policy, model identity, quantization scheme, architecture parameters, and deployment specs are committed and checked against known-good values.
2. **Shell verification (exact).** Information-theoretically sound Freivalds checks ($\leq 1/p$ false-accept in the current field) on all shell weight matmuls, plus deterministic or canonical recomputation of the exact INT8 bridge tensors, Q/K/V dequantization, bias handling, RoPE, RMSNorm, and SiLU.
3. **KV provenance (statistical by default).** Merkle-committed per-token K,V history; sampled positions are shell-verified. Binding is exact; correctness of unsampled positions depends on sampling rate unless deep audit is used.
4. **Cross-layer consistency (structural).** Opening multiple layers on the same token creates algebraic coupling through the residual stream, so fake attention must stay consistent across all opened layers.
5. **Attention input audit (audited but not verified).** Arbitrary-position attention outputs are explicitly out of scope. The verifier audits attention inputs and wiring only: score anchoring, KV provenance, GQA / RoPE-config / causal-mask wiring, and a token-0 local replay smoke check (Section 6).
6. **Final-token replay (exact / fail-closed).** The verifier starts from the captured pre-final-norm residual, checks the LM-head boundary with Freivalds, and replays the decode/output policy. Sampled decode is verified exactly via captured GPU logits plus the LM-head Freivalds binding; greedy decode may use cheaper validated logits paths where explicitly supported. Unsupported semantics are rejected explicitly rather than silently accepted.

Table 2 traces the full forward pass for one output token. Each operation is labeled with its verification type: Freivalds-checked weight matmuls, canonically recomputable bridge operations, and the single point that the protocol does not verify: FP16/BF16 attention, which is instead bracketed by audits on its inputs and wiring. The only non-replayable provider-side state is the per-layer `attn_out_i8` and its quantization scale, plus the captured pre-final-norm residual that anchors the exact final-token tail.

```
residual = embedding[token_id]                          exact lookup
for each layer:
  x_norm = RMSNorm(residual)                            canonical
  x_i8, s = quantize(x_norm)                            exact
  q_i32 = W_q @ x_i8                                    Freivalds
  k_i32 = W_k @ x_i8                                    Freivalds
  v_i32 = W_v @ x_i8                                    Freivalds
  q = RoPE(add_bias(dequant(q_i32, ...)),               canonical
pos)
  k = RoPE(add_bias(dequant(k_i32, ...)),               canonical
pos)
  v = add_bias(dequant(v_i32, ...))                     canonical
  attn = softmax(q @ k^T / sqrt(d)) @ v                 audited inputs
  attn_out_i8, sa = quantize(attn)                      STORED
  o_i32 = W_o @ attn_out_i8                             Freivalds
  residual += dequant(o_i32, ...)                       canonical
  x_norm = RMSNorm(residual)                            canonical
  x_i8, s = quantize(x_norm)                            exact
  gate_i32 = W_gate @ x_i8                              Freivalds
  up_i32   = W_up   @ x_i8                              Freivalds
     x   =   SiLU(dequant(gate_i32))   *                canonical
dequant(up_i32)
  x_i8, s = quantize(x)                                 exact
  down_i32 = W_down @ x_i8                              Freivalds
  residual += dequant(down_i32, ...)                    canonical
final_residual = residual                               STORED
x_norm = RMSNorm(final_residual)                        canonical
x_i8, s = quantize(x_norm)                              exact
logits boundary = LM_head(x)                            Freivalds
token = decode(bound logits, seed, policy)              exact / fail-closed
```

Table 2: Annotated forward pass for one output token. Each operation is labeled with its verification type. The only stage the protocol does not verify is FP16/BF16 attention; instead the verifier audits its inputs (score anchor, KV provenance) and wiring (GQA, RoPE config, causal mask). The final-token tail is exact because it starts from the captured pre-final-norm residual.

Component	**Verification method**	**Guarantee**	**What is bound**
Input preprocessing	Committed input spec	Exact	Tokenizer, chat template, BOS/EOS, truncation, special-token handling, system prompt
Embedding	Merkle proof to committed embedding root	Exact	Token-to-row binding
Shell matmuls	Freivalds	Exact	Seven shell matrix families
Bridge operations	Canonical recomputation	Exact	Requantization, residual, RMSNorm, RoPE, SiLU
Prefix / KV provenance	Merkle binding + sampled shell checks or deep audit	Statistical / Exact	Committed prefix state
Attention (arbitrary position)	Out of scope; not verified	Audited inputs only	—
Score anchor (last gen. token)	Recompute $QK^T/\sqrt{d}$ vs witnessed scores	Audited input	Witnessed pre-softmax scores
KV provenance (attention input)	Opened K/V rows match committed cache rows	Audited input	Cache-row provenance
Wiring (GQA / RoPE / mask)	Config-hash + structural mask check (mask is last-gen.-token)	Audited input	GQA layout, RoPE config, causal mask
Local replay smoke (token 0)	Exact replay at token 0	Regression check	Token-0 attention only
Final boundary	Captured pre-final-norm residual	Exact	Start of the exact final-token tail
LM head	Freivalds binding; exact replay where supported	Exact	Final linear map / captured logits boundary
Sampled decode	Captured GPU logits + LM-head Freivalds binding	Exact	Sampled token identity
Decode policy	Canonical replay or explicit rejection	Exact / Fail-closed	Sampler semantics and randomness
Output policy	Exact replay or explicit rejection	Exact / Fail-closed	Stopping and text-cleanup semantics

Table 3: Verification coverage matrix for the final protocol. "Fail-closed" means a feature is either replayed exactly or rejected explicitly; the verifier never silently accepts unsupported semantics.

4 The Protocol

4.1 Phase 0: Setup

Public identity. A Merkle root [6] R_W is computed over all weight matrices of the published checkpoint. This root is the model's public identity; anyone can recompute it from the published weights.

Verifier key generation. The verifier fixes a prime field modulus p; in the current implementation $p = 2^{32} - 5$. All Freivalds checks operate in $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$: INT8 inputs and INT32 accumulators are lifted into this field. For the 7 shell matrix types (W_q, W_k, W_v, W_o, W_{gate}, W_{up}, W_{down}) and the final LM head, the verifier samples secret random vectors r_j uniformly from $\mathbb{F}_p^m$. For each per-layer matrix, the verifier precomputes $v_j^{(i)} = r_j^T W_j^{(i)} \bmod p$; for the LM head, the verifier precomputes the corresponding verifier-side check vector once for the global unembedding matrix. The verifier key also binds the embedding commitment, final RMSNorm weights, model configuration, quantization metadata, RoPE/scaling configuration, and RMSNorm epsilon needed for canonical replay. In the current benchmark projection this verifier key is roughly ~ 48 MB for Llama 70B, still small enough for client-held verification. After precomputation, the verifier deletes the weights. The r_j vectors are verifier-secret; if the provider learns them, it can forge passing checks. Freivalds is therefore used here as verifier-secret randomized algebraic verification inside the broader cryptographic protocol, not as a standalone public proof system.

Specification binding. The deployment is described by four committed specs: H_{input}, H_{model}, H_{decode}, and H_{output}. These bind preprocessing semantics, model configuration, decode policy, and output policy respectively. Unsupported semantics are not left ambiguous: they must either be replayed exactly or rejected explicitly.

4.2 Phase 1: Commitment

The provider runs inference normally. The serving path is unchanged except for a tracing layer that captures intermediates alongside execution. In batched serving (e.g., vLLM with paged attention [7]), multiple requests share GPU resources; the tracing layer captures per-request intermediates, extracting per-request activations from the batched computation without modifying the batched kernels.

For every token at every layer (both prefill and decode), the provider records:

- The INT8 input to each of the 7 weight matrices
- The INT32 accumulator output ($W_{\text{i8}} \times x_{\text{i8}}$, exact)
- The requantized INT8 values at each bridge
- The post-attention INT8 output
- The per-tensor quantization scales at each requantization bridge
- The captured pre-final-norm residual needed to anchor the exact final-token tail

After generation completes, the provider builds two Merkle trees [6]. The trees are separate because they serve different access patterns: shell verification (Step 2) opens a few positions from R_T, while KV provenance (Step 3) opens the full prefix from R_{KV}. A single tree would force opening all intermediates at every prefix position just to extract the KV values.

- R_T (trace tree): over all intermediates at all tokens. The provider cannot change any activation after committing.
- R_{KV} (KV tree): over per-token K, V state across all layers. The provider cannot retroactively rewrite earlier tokens' context. This tree allows efficient opening of the full prefix KV without opening the complete trace at every position.

A deployment commitment binds everything outside the forward pass:

$$M = H\big(H_{\text{input}} \parallel H_{\text{model}} \parallel H_{\text{decode}} \parallel H_{\text{output}}\big) \tag{1}$$

The provider also samples a fresh per-request transcript seed s, commits $H(s)$ in the receipt, and derives per-token randomness deterministically from s and the token index.

The provider returns the response plus a compact receipt $(R_T, R_{\text{KV}}, M, H(s), N, ...)$. Most responses are never audited. The only normal-path additions are the tracing layer and receipt machinery; the heavier Merkle/packing finalization can run after generation completes.

4.3 Phase 2: Verification

The client decides to audit a response. The provider does not know which responses will be challenged or which tokens within them will be opened.

4.3.1 Step 1: Challenge

The verifier selects c random token positions from the N-token response and, for each, chooses ℓ layers to open. Opening multiple layers on the same token is preferred, since it enables cross-layer consistency checks (Step 4).

4.3.2 Step 2: Shell Verification

For each challenged token t at each opened layer i, the provider opens the INT8 inputs x_{i8}, INT32 accumulators z_{i32}, per-tensor quantization scales, the residual stream values at the layer boundaries, and the post-attention output attn_out_i8, all with Merkle proofs against R_T. The verifier performs four checks:

1. **Freivalds on each shell weight matrix.** For each of the 7 shell matrices, the verifier checks $v_j^{(i)} \cdot x \equiv r_j^T \cdot z \pmod p$. Each check is two dot products in $\mathbb{F}_p$, an $O(n)$ operation. If the provider used wrong weights, false-accept probability is $\leq 1/p$ per matrix. The bound follows from the Schwartz–Zippel lemma: a nonzero linear form over $\mathbb{F}_p$ vanishes on at most a $1/p$ fraction of inputs.
2. **Bridge tensors (exact).** Wherever an exact INT8 tensor feeds a later INT8 stage (for example the committed x_{attn} boundary and the FFN bridge tensors), the verifier recomputes the canonical bridge from the opened accumulators, quantization scales, and residual-stream state. This prevents the provider from passing Freivalds on the INT32 accumulators while feeding fabricated INT8 bridge values into downstream checks. Q/K/V themselves are not separately bridged through committed $q_{\text{i8}}/k_{\text{i8}}/v_{\text{i8}}$ values in the canonical path; they are reconstructed by dequantization, bias application, and RoPE from the opened accumulators. The retained post-attention output attn_out_i8 is **not** re-derived here, because arbitrary-position attention outputs are out of scope; the attention input audit (Step 5) constrains Q, K, V, witnessed scores, and wiring instead.

3. **SiLU (exact / canonical).** In the toy path, INT8 inputs permit a 256-entry lookup table. In the production W8A8 path, the verifier canonically recomputes the scaled SiLU bridge from dequantized gate/up accumulators and checks the requantized result.
4. **RoPE and RMSNorm.** RoPE is canonically recomputed from the position index and model configuration in deterministic f64 arithmetic. RMSNorm is canonically recomputed from the opened residual stream values; the result is verified in the quantized output space (the recomputed value must requantize to the same i8 as the committed value).

After this step, the verifier has trusted Q_t, K_t, V_t for the challenged token as shell-verified accumulator outputs together with their canonical dequantization, bias handling, and RoPE reconstruction.

4.3.3 Step 3: KV Provenance

Attention at token t requires the full prefix $K_{1..t}$, $V_{1..t}$. Shell verification gives trusted values for the challenged token but not the prefix.

For each opened layer i, the provider opens the K, V values at all prefix positions $j < t$ with Merkle proofs against R_{KV}. The verifier then:

1. Verifies the Merkle proofs, confirming these are the values the provider committed in Phase 1.
2. Randomly samples k earlier token positions $j_1, ..., j_k < t$.
3. For each sampled position j_s, runs full shell verification at layer i (Freivalds on all 7 matrices, requantization, RoPE, RMSNorm, SiLU), producing independently verified $K_{j_s}^{(i)}$, $V_{j_s}^{(i)}$.
4. Compares these shell-verified values against the corresponding entries opened from R_{KV}.

If the provider tampered with f out of n prefix positions and the verifier samples k:

$$P(\text{catch}) = 1 - (1 - f/n)^k \tag{2}$$

Commitment binding is exact (hash collision resistance). Correctness of unsampled positions is statistical: the provider risks the verifier sampling any corrupted position.

4.3.4 Step 4: Cross-Layer Consistency

There is no separate proof object here; the consistency check comes from opening multiple layers and verifying the committed bridges between them. When the verifier opens layers L and $L+1$ on the same token, fake attention at layer L must produce a post-attention output that, after requantization and W_o, feeds into layer $L+1$'s RMSNorm consistently with the committed trace. Both sides of this boundary are shell-verified, so the adversary cannot fabricate a consistent bridge without matching the committed values exactly. The more layers opened, the tighter the constraint. In full-audit mode (all layers opened), fake attention at every layer must stay mutually consistent through the entire residual stream.

4.3.5 Step 5: Attention Input Audit

Arbitrary-position attention outputs are explicitly **not** verified in the kept protocol. Every attempted production verification path for the attention interior (exact stock-kernel replay, tiled / LSE replay, stock-bounded certification on FP16↔FP64 corridors, and deterministic kernels) failed to give a tight, kernel-portable, production-cost claim on stock GPU FlashAttention [8]. The honest decomposition is therefore to bracket the attention block by audits on its **inputs** and **wiring** and leave the interior unverified.

This step combines the trusted Q_t reconstructed in Step 2, the commitment-verified prefix $K_{1..t}$, $V_{1..t}$ from Step 3, and additional witness data captured by the prover during execution. The verifier runs four sub-audits.

1. **Score anchor.** The verifier recomputes $Q_t \cdot K_j/\sqrt{d}$ for all prefix positions j in deterministic f64 arithmetic from the shell-verified Q_t and the committed $K_{1..t}$, and compares against the prover-side **witnessed pre-softmax scores** committed in R_T. The maximum absolute element-wise gap is reported as evidence; in audit-only mode it is not used as a hard verification gate. **Witness scope.** Under the current witness contract, the prover retains Q for the **final decode step only**, so the score-anchor sub-audit applies only when the audited token is the last generated token ($t = N-1$). Per-step Q retention (full per-position score audit) is a deferred extension; see Section 9.
2. **KV provenance.** For each opened layer i, the provider opens the K, V values at all prefix positions $j < t$ with Merkle proofs against R_{KV} (Step 3). Step 5 additionally checks that the opened cache rows match the committed token-position / cache-row mapping and respect page boundaries: the verifier checks structural cache-row provenance, and any mismatch fails the audit. This applies to the full opened range, not only the last generated token.
3. **Wiring (GQA / RoPE-config / causal mask).** The verifier checks three wiring properties: (a) GQA head mapping (number of query heads, number of K/V heads, and per-head dimension) matches the verifier-key configuration; (b) RoPE configuration is bound by hashing `rope_theta` and any `rope_scaling` block (e.g., Llama 3 long-context scaling) and comparing to the verifier-key hash; (c) the causal mask is structurally valid. The mask sub-audit is derived from the witnessed pre-softmax scores (same witness as the score anchor) and therefore inherits the same last-generated-token scope; GQA and RoPE-config audits are independent of token index.
4. **Local replay smoke (token 0).** As a regression check only, the verifier runs an exact replay at token 0 (where Q has no prefix and softmax reduces to a one-hot). This is **not** a product attention claim and is not extended to arbitrary positions.

This bracket pins down attention's inputs and wiring, but does not pin the attention output. The adversary could in principle fabricate a different post-attention output, as long as (a) it survives cross-layer consistency through W_o and the residual stream into later opened layers (Step 4), (b) the prover-side score witness it commits is consistent with $Q_t \cdot K_j/\sqrt{d}$ on the audited token, and (c) the surrounding wiring audits pass. The product claim is correspondingly narrower: stock-mode attention is **audited but not verified**, and any quantitative downstream argument must rest on cross-layer consistency, the input/wiring audits, and the rest of the exact pipeline (decode, LM head, sampled-token capture).

Limitations. The audit constrains consistency with the **committed** prefix, not necessarily with true execution at every earlier token; prefix KV values are commitment-verified but only statistically anchored to real computation via the sampled shell checks in Step 3. The score-anchor and causal-mask sub-audits are also bound by the current witness scope (last generated token only) and would require per-step Q retention to extend to arbitrary generated positions.

4.3.6 Step 6: Final-Token Replay

The verifier now enters the exact final-token tail. The provider opens the captured pre-final-norm residual for the challenged token, together with the committed decode/output policy, the revealed transcript seed, and (in the kept sampled-decode path) the captured GPU logits used by the sampler. The verifier then:

1. Applies the final RMSNorm exactly using the committed model spec.
2. Quantizes into the LM-head input space canonically.
3. Checks the LM-head boundary with Freivalds.
4. For **sampled** decode, additionally Freivalds-binds the captured GPU logits back to `lp_hidden × lm_head_bf16`, then verifies that the sampler applied to those captured logits with the per-token seed reproduces the emitted token exactly. For **greedy** decode, cheaper validated paths (i32 exact-token-identity or LP-hidden bf16) remain acceptable where explicitly validated.
5. Replays the canonical decode policy using the committed decode spec and the per-token seed derived from the revealed transcript seed.
6. Verifies the chosen token and the output-policy behavior (EOS handling, stop strings, max/min stopping rules, and claimed text cleanup) exactly, or rejects the feature explicitly if the deployment claims an unsupported policy.

This step is what makes the final-token boundary exact. Sampled decode is verified exactly via the captured-logits binding, not derived from a hidden state reconstructed through many layers of unverified attention output.

4.4 Summary

Table 4 summarizes the verification method applied to each operation in the forward pass.

Operation	**Verification**
Input / deployment specs	Committed four-spec manifest
Input embedding	Table lookup (exact)
W_q, W_k, W_v (INT8)	Freivalds
INT8 bridge tensors (x_{attn}, x_{ffn})	Exact recomputation
`attn_out_i8` (committed post-attention output)	Retained as a commitment input; not re-derived
Q, K, V dequant + bias + RoPE	Canonical recomputation
Attention output (arbitrary position)	Out of scope; not verified
Score anchor (last gen. token)	Recompute $QK^T/\sqrt{d}$ vs witnessed scores
KV provenance (attention input)	Cache-row provenance vs commitments
Wiring (GQA / RoPE / mask)	Config-hash + structural mask (mask is last-gen.-token)
Local replay smoke (token 0)	Exact replay at token 0 only
W_o (INT8)	Freivalds
RMSNorm	Canonical recomputation
W_{gate}, W_{up} (INT8)	Freivalds
SiLU $\odot$ up	Canonical recomputation + exact requantized check
W_{down} (INT8)	Freivalds
Final boundary	Captured pre-final-norm residual
LM head	Freivalds binding; exact replay where supported
Sampled decode	Captured GPU logits + LM-head Freivalds binding
Decode policy	Canonical replay or fail-closed rejection
Output policy	Exact replay or fail-closed rejection

Table 4: Per-layer verification methods

4.5 Audit Walkthrough: Last Generated Token, Two Layers

Audit the last generated token $t = N - 1$ of a 70B model at layers 12 and 13, sampling $k = 8$ prefix positions for KV provenance.

1. **Challenge.** Verifier sends $(t = N - 1, \{12, 13\})$. The last-generated-token choice is required by the current witness contract for the score anchor and causal-mask sub-audits; KV provenance and the GQA / RoPE-config wiring sub-audits do not depend on token choice.

2. **Shell.** Provider opens trace at both layers from R_T. Verifier runs Freivalds on the applicable shell matrices per layer, recomputes all bridges. Yields trusted Q_t, K_t, V_t at both layers.
3. **KV provenance.** Provider opens $K_{1..t-1}$, $V_{1..t-1}$ at both layers from R_{KV}. Verifier picks 8 random earlier positions, runs full shell verification at each, confirms committed K, V match, and additionally checks structural cache-row / page-boundary provenance over the full opened range.
4. **Cross-layer.** Post-attention output at layer 12 feeds into layer 13′s RMSNorm; both sides are shell-verified, so the residual-stream boundary must match exactly.
5. **Attention input audit.** At each opened layer, the verifier (a) recomputes $Q_t \cdot K_j/\sqrt{d}$ in deterministic f64 for all prefix positions j and compares against the prover-side witnessed pre-softmax scores (max element-wise gap reported as evidence; not a hard gate); (b) checks the structural causal mask against the witnessed scores; (c) verifies GQA head mapping and the RoPE-config hash. Arbitrary-position attention outputs are not verified at any layer.
6. **Final token (exact).** Provider opens the captured pre-final-norm residual, the captured GPU logits, and the revealed transcript seed. Verifier applies final RMSNorm exactly, checks the LM head with Freivalds, exactly binds the captured GPU logits via the LM-head check, replays the canonical decode policy on those logits, and confirms the chosen token and output policy.

4.6 Data Lifecycle

Table 5 shows what data exists at each stage of the protocol.

Receipt fields (compact)	Provider retains (ring buffer)	Revealed on audit
R_T (trace Merkle root)	INT8 inputs to all 7 matrices	Merkle openings from R_T
R_{KV} (KV Merkle root)	INT32 accumulator outputs	Merkle openings from R_{KV}
Spec commitment M	Requantized INT8 at each bridge	Opened intermediates at challenged positions
Seed commitment $H(s)$	`attn_out_i8` + quantization scales	Full prefix K, V at opened layers
Token count N	Captured pre-final-norm residual	Opened final boundary state + revealed seed
	Captured GPU logits (per generated token)	Witnessed pre-softmax scores at last gen. token
	Per-tensor quantization scales	Captured GPU logits at challenged token

Table 5: Data lifecycle: receipt (every response), retained state (RAM ring buffer, discarded after audit window), and audit openings (revealed only when challenged).

5 Security Analysis

5.1 Security Game

We formalize model-identity soundness via an interactive game between a challenger $\mathcal{C}$ (verifier) and an adversary $\mathcal{A}$ (provider).

Game$^{\text{id}}_{\mathcal{A}}(\lambda)$:

1. **Setup.** $\mathcal{C}$ publishes weights W and Merkle root R_W. $\mathcal{C}$ samples verifier-secret Freivalds vectors for every shell matrix family and for the LM head, precomputes the corresponding verifier-side checks, and stores the secret verifier key sk. $\mathcal{A}$ receives W, R_W but never observes sk.
2. **Commit.** $\mathcal{A}$ receives a prompt, runs inference (using any strategy), and returns a response plus receipt $(R_T, R_{\text{KV}}, M, H(s), N, ...)$.
3. **Challenge.** $\mathcal{C}$ selects c token positions and ℓ layers per token uniformly at random. $\mathcal{A}$ learns the challenge only now.
4. **Open.** $\mathcal{A}$ opens the trace at challenged positions with Merkle proofs against R_T.
5. **Verify.** $\mathcal{C}$ runs Freivalds and bridge checks at every opened position. Outputs accept or reject.

Definition
(Model-identity soundness). $\text{Adv}^{\text{id}}_{\mathcal{A}} = \Pr\big[\text{Verify accepts} \wedge \mathcal{A} \text{ used } W'_j \neq W_j \text{ at any Freivalds-checked matrix}\big]$. The protocol is ε-sound if $\text{Adv}^{\text{id}}_{\mathcal{A}} \leq \varepsilon$ for all adversaries $\mathcal{A}$.

5.2 Model Swap or Downgrade

Proposition 1 (Model identity). *For each Freivalds-checked matrix, if the provider used $W_j'^{(i)} \neq W_j^{(i)}$, the Freivalds check rejects with probability $\geq 1 - 1/p$.*

Proof sketch. The adversary commits (x, z) via R_T before learning r_j. If $z \neq W_j^{(i)} x$ (wrong weights), let $d = W_j^{(i)} x - z \neq 0$. The check $v_j^{(i)} \cdot x \equiv r_j^T \cdot z \pmod p$ reduces to $r_j^T \cdot d \equiv 0 \pmod p$. Since $d \neq 0$ is fixed before r_j is revealed, and r_j is uniform over $\mathbb{F}_p^m$, exactly p^{m-1} of p^m vectors satisfy this linear equation. Thus $\Pr[\text{accept}] = 1/p$ (about $1/(2^{32} - 5)$ in the current implementation). □

A single audit of a single token suffices. The guarantee is unconditional: it does not depend on statistical sampling or threshold calibration.

5.3 Attention Manipulation

Proposition 2 (Attention input audit). *Arbitrary-position attention outputs are not verified. If the audit-only stock-mode attention path accepts, then on every audited (token, layer):*

(i) the prover-side witnessed pre-softmax scores agree with the canonical $Q_t \cdot K_j/\sqrt{d}$ computed in f64 from the shell-verified Q_t and the committed $K_{1..t}$ on the audited token (with score-anchor and causal-mask audits scoped to $t = N - 1$ under the current witness contract);

(ii) opened K/V cache rows match committed token-position / cache-row mappings on the full opened range;

(iii) GQA head layout and the RoPE configuration hash match the verifier-key configuration.

The shell locks the inputs (Q, K, V) and the output projection (W_o) of every attention block. Cross-layer consistency (Step 4) further forces any fabricated post-attention output to survive W_o and the residual stream into all opened later layers. Within those

constraints, the audit pins down the **score** (against witnessed scores), the **KV provenance**, and the **wiring**, but it does not pin down the attention output itself. The product claim is that stock-mode attention is **audited but not verified**: an adversary who fakes the attention output must additionally produce a witness-score story consistent with $QK^T/\sqrt{d}$ and survive W_o + the cross-layer residual binding into later opened layers and the exact final-token tail.

5.4 Fabricated Prefix Context

Proposition 3 (KV tampering detection). *If the provider tampered with f out of n prefix positions and the verifier samples k, detection probability is* $P(\text{catch}) = 1 - (1 - f/n)^k$.

Committed KV values cannot change after R_{KV} is sent (hash collision resistance). The verifier anchors the commitment to real computation by sampling earlier positions and running full shell verification. Detection probability increases monotonically with the tampering fraction f/n.

5.5 Deployment Configuration Tampering

The deployment commitment M binds four specs into the receipt: preprocessing, model configuration, decode policy, and output policy. This includes the tokenizer, chat template, BOS/EOS preprocessing policy, truncation semantics, weight identity, quantization scheme, RoPE/scaling configuration, RMSNorm epsilon, adapter identity, sampler version, temperature/top-k/top-p, penalties, grammar constraints, stop policy, and detokenization semantics. The verifier checks these against known-good values for the intended deployment. Any change to the deployment configuration produces a different committed spec surface.

5.6 Assumptions Outside Protocol Scope

The protocol does not assume honest provider hardware or honest provider runtime behavior. Those are the target of verification. The explicit assumptions outside protocol scope are:

1. standard cryptographic assumptions for the hash functions
2. information-theoretic soundness of the finite-field Freivalds checks
3. secrecy of verifier-only material such as the Freivalds vectors
4. no side-channel leakage of verifier-secret material
5. correct execution of the verifier itself

6 The Attention Gap

The shell is exactly verifiable because its operations are deterministic or canonically recomputable: INT8 matmuls are checked with information-theoretically sound Freivalds checks, while requantization, RoPE, SiLU, and RMSNorm are verified by exact or canonical recomputation. Attention (QK^T, softmax, αV) is computed in FP16/BF16, which is not bit-reproducible across hardware. Every attempted production verification path for this interior (exact stock-kernel replay, tiled / LSE replay, stock-bounded certification on FP16↔FP64 corridors, and deterministic kernels) failed to give a tight, kernel-portable, production-cost claim on stock GPU FlashAttention. Earlier corridor measurements are reported below as background evidence about how stock FP16/BF16 attention behaves; they are **not** a protocol guarantee, since arbitrary-position attention outputs are no longer claimed to be verified.

In the kept protocol, the attention interior is therefore **not** verified directly. The protocol constrains it from the input side instead. The shell exactly verifies Q, K, V at challenged positions and W_o on every layer. Step 5 (Section 4.3.5) then runs four sub-audits on the attention block:

1. **Score anchor** against witnessed pre-softmax scores (last-generated-token only under the current witness contract).
2. **KV provenance** against committed cache rows (full opened range).
3. **Wiring** (GQA layout, RoPE config hash, structural causal mask, with the mask sub-audit sharing the score-witness scope).
4. **Local replay smoke** at token 0 only, as regression check.

Cross-layer consistency (Step 4) further forces any fabricated post-attention output to survive W_o and the residual stream into all opened later layers, and the exact final-token tail (Step 6, including captured-logits sampled-decode binding) closes the back end.

The remaining adversarial freedom comes from three sources:

1. **Unverified attention output.** Arbitrary-position attention outputs are out of scope. An adversary's freedom here is only constrained by (a) cross-layer consistency through W_o and the residual stream into opened later layers, (b) the score-anchor sub-audit on the last generated token, (c) the structural mask / wiring audits, and (d) the exact captured-logits sampled-decode binding at the final-token boundary. Translating these constraints into an explicit downstream output bound is open work.
2. **Witness scope.** Score anchoring and the causal-mask sub-audit currently apply only to the last generated token, because the prover-side witness retains Q for the final decode step only. Extending these audits to arbitrary generated positions requires per-step Q retention; KV provenance and the GQA / RoPE-config sub-audits already cover the full opened range and are independent of token index.
3. **Statistical KV anchoring.** The prefix K, V values used in the score anchor are commitment-verified (they match R_{KV}) but only statistically anchored to real computation via sampled shell checks. Unsampled prefix positions are not independently verified, so the score-anchor sub-audit attests consistency with the *committed* prefix, not necessarily with the true execution at every earlier token.

Model	Wklds	Max pos.	Max L_{inf}	1st-gen	Frac. ≤ 1
Qwen2.5-7B W8A8	6	1164	8	5	$> 99.8\%$
Llama-3.1-8B W8A8	6	1165	9	5	$> 99.9\%$

Table 6: Background evidence on the FP16↔FP64 corridor for stock GPU attention on Qwen2.5-7B-W8A8 and Llama-3.1-8B-W8A8. Reported here as context only; the kept protocol does not verify arbitrary-position attention outputs and therefore does not depend on this corridor as a security claim. Measurements assume architecture-correct replay semantics (Q/K/V bias handling, model-specific RoPE conventions/scaling) and were collected on A100-80GB GPUs under vLLM eager mode.

7 Provider Costs

7.1 Latency

We separate two costs. **Online generation overhead** is the user-visible slowdown from trace capture, hooks, and synchronization while tokens are being generated and streamed. **Post-generation finalization overhead** is the additional cost to build and pack the receipt after generation has completed. The former is the main serving-path metric; the latter matters for provider throughput and infrastructure sizing, but not for time-to-first-token or live streaming latency if receipt finalization is asynchronous.

Across the currently measured configurations, online overhead is stable at roughly 12–14% (about 1.3–1.8 ms/token) from Qwen2.5-7B-W8A8 through Llama-3.1-70B-W8A8. These serving-overhead measurements were collected on A100-80GB GPUs under vLLM eager mode. Post-generation finalization is currently much larger, but it occurs after generation and is therefore off the critical user-facing path. Profiling indicates that the dominant post-generation cost in the current prototype is the Rust-side commitment routine, especially trace hashing, Merkle construction, and eager KV transcript derivation; Python-side preparation is negligible by comparison.

Model	Cfgs	Tokens	Online OH	Full packed OH
Qwen2.5-7B W8A8	4	105–297	12.7–13.8%	225–338%
Llama-3.1-8B W8A8	2	104–168	13.5–13.7%	422–554%
Llama-3.1-70B W8A8	1	296	12.1%	390%

Table 7: Measured serving overhead on the current implementation. "Online OH" is the user-visible generation-path overhead from capture, hooks, and synchronization. "Full packed OH" includes post-generation receipt finalization and packing. The current matrix is complete for Qwen2.5-7B-W8A8 and partial for the two Llama models because the longer 70B finalization runs hit the benchmark timeout.

7.2 Verifier Cost

Client-side verification runs entirely on CPU. We benchmarked the per-token verification cost at real model dimensions using random data (the verifier needs only the precomputed key and the opened trace, not the actual weights). Measurements on a commodity laptop CPU (Apple M4):

Model	Layers	10 layers	All layers	Per layer
Llama 3.1 8B	32	0.66 ms	1.98 ms	62 µs
Llama 3.1 70B	80	1.27 ms	10.1 ms	126 µs
Llama 3.1 405B	126	2.33 ms	29.8 ms	236 µs

Table 8: Measured per-token verification cost (Freivalds checks on all 7 matrices per layer + SiLU recomputation). "10 layers" is the routine stratified audit; "All layers" is the full audit. Merkle and IO-chain proof checks add <1 µs and are omitted. For S challenged tokens, multiply by S.

For the default audit mode ($S = 1$ token, 10 routine layers), verifying one Llama 70B response takes about 1.3 ms, well within CPU-feasible bounds. Even a full-audit pass over all 80 layers at 5 challenged tokens costs roughly 50 ms.

7.3 Storage

Only the non-derivable retained trace state requires storage: post-attention INT8 outputs, the associated per-tensor quantization scales, and the captured pre-final-norm residual that anchors the exact final-token tail (everything else is re-derivable):

This storage accounting covers only retained per-token trace state. It excludes sampled-decode evidence, which is stored separately when the `CapturedLogits` decode path is enabled. The deployment commitment M is computed and bound into the receipt at commit time; it is configuration metadata, not non-derivable trace state.

Model	Layers	hidden_dim	Per token
Llama 8B	32	4096	∼ 144 KB
Llama 70B	80	8192	∼ 672 KB
Llama 405B	126	16384	∼ 2.1 MB

Table 9: Per-token storage for non-derivable intermediates

Sampled decode adds a separate retained artifact: the exact GPU logits used by the sampler. These logits are committed for every generated token because the audit challenge is chosen after generation, but they are opened only for challenged tokens. On the measured captured-logits profiles, retained sampled-decode state is about 501 KiB/token for Llama and 594 KiB/token for Qwen, and audit bandwidth increases by roughly 0.5–0.6 MiB per opened token.

Profile	Retained logits	Opened payload
Llama W8A8 captured-logits	∼ 501 KiB/token	∼ 0.5 MiB/challenged token
Qwen W8A8 captured-logits	∼ 594 KiB/token	∼ 0.6 MiB/challenged token

Table 10: Additional sampled-decode retention for the captured-logits path. This is separate from the trace-state storage in the previous table.

7.4 Audit Window

With a short audit window (1–2 minutes), traces fit in a RAM ring buffer with no disk I/O. For Llama 70B on 4× H100, assuming an aggregate trace-state write rate of ∼ 1.3 GB/s, a 2-minute window requires ∼ 156 GB (∼ 8% of 2 TB system RAM). Captured-logits sampled decode adds ∼ 0.5 MiB per generated token of retained logits on top of the ∼ 672 KB/token trace state, raising the aggregate rate to ∼ 2.3 GB/s and the 2-minute window to ∼ 270 GB (∼ 14%). Longer windows spill to NVMe or networked storage.

Short audit windows require automated auditing: the client's audit decision must be programmatic.

8 Related Work

The most direct alternative to CommitLLM is proving inference in zero knowledge. Earlier verifiable-inference systems such as SafetyNets [2] and later zk approaches such as zkCNN [3] show

that inference correctness can be proved by reducing neural networks to arithmetic-circuit-style verification. However, current ZK systems still impose large prover overheads. For a model that costs dollars per response to serve, this translates to hundreds or thousands of dollars per proved response, prohibitive for production inference. The core design difference is economic: CommitLLM keeps the provider on the normal serving path and keeps the verifier CPU-feasible, while a SNARK-style design pushes large extra cost into per-response proof generation in exchange for a compact transferable proof.

At a more foundational level, CommitLLM also sits in the lineage of interactive proofs and delegated computation. Classical interactive-proof formulations [9] and later delegated-computation protocols [10] study how a verifier can check outsourced computation without performing the full work locally. CommitLLM does not instantiate those general-purpose protocols directly; instead, it specializes the verification structure to open-weight INT8 LLM inference, using public weights, commitment binding, and verifier-secret algebraic checks to obtain a much cheaper normal serving path.

Related cryptographic ML systems such as Delphi [11] address a different problem: privacy-preserving neural inference, where neither party should reveal its inputs or model. CommitLLM instead targets model-provenance auditing for open weights, where the model is public and the goal is to verify that the claimed weights were actually used.

CommitLLM makes the opposite tradeoff: verification is interactive (the provider must respond to challenges) and the receipt alone is not a succinct SNARK-like transferable proof. In exchange, the normal serving path remains close to ordinary serving: the measured online overhead from tracing is currently about 12–14%, while heavier receipt finalization is shifted after generation. Transferability exists only in a weaker disclosed-audit sense: a fully disclosed audit transcript can be re-checked by third parties, but that is a bulky audit bundle rather than a compact proof. The expensive part, verification, only occurs on the small fraction of responses that are audited.

Trusted execution environments (TEEs) and remote attestation provide another alternative [4], [12]. TEEs can attest that specific code ran on specific hardware, avoiding the overhead of proof systems, but they introduce hardware trust assumptions, limit deployment flexibility, and tie verification to a specific vendor's attestation chain. CommitLLM requires no hardware trust beyond standard computation.

Approach	**Provider overhead**	**Guarantee**	**Trust assumption**	**Transf.**
ZK proofs	100–1000 × prover	Full computation correct	None (math)	Yes
TEEs	Attestation HW	Claimed code ran on attested HW	Hardware vendor	Yes
Redundant exec.	2–3× compute	Outputs agree across replicas	Honest majority	No
CommitLLM	∼12–14% online + post-gen finalization + rare audits	Claimed weights used (exact); sampled decode exact; attention inputs/ wiring audited (arbitrary-position outputs not verified)	Verifier key secrecy + verifier integrity	Audit bundle

Table 11: Comparison of verifiable inference approaches.

9 Limitations and Extensions

The kept protocol does **not** verify arbitrary-position attention outputs. Every attempted production path for that interior was closed (exact stock-kernel replay, tiled / LSE replay, stock-bounded certification on FP16↔FP64 corridors, deterministic kernels), so the honest claim is the audit-only stock-mode attention path described in Section 4.3.5. The remaining open questions are not whether a corridor exists (corridor measurements appear in Section 6 as background only) but how much the audit-only bracket can be tightened, what downstream output bound it actually implies through W_o and the residual stream, and what audit/storage policy providers will accept at production scale. Closing the gap to verified arbitrary-position attention would require either deterministic attention kernels, which change the serving path and violate the sidecar design constraint, or an interactive proof of the attention computation generated only for challenged tokens at audit time. The latter is compatible with the sidecar design but unexplored; we leave it as future work. The final protocol also assumes an autoregressive decoder-only architecture with the committed capture layout and architecture-correct replay semantics; broader architecture support requires additional schema and replay work but does not change the guarantee taxonomy.

A full composed soundness theorem remains future work. The current protocol is commitment-bound end-to-end: large linear components are verified by verifier-secret, information-theoreti-

cally sound algebraic checks; supported nonlinear components by canonical replay; sampled decode exactly via captured GPU logits plus the LM-head Freivalds binding; stock-mode attention is **audited but not verified** (score anchoring, KV provenance, GQA / RoPE-config / causal-mask wiring, plus token-0 local replay smoke); and routine KV provenance is statistical unless deep audit is used.

The residual attention hole is concrete, not merely semantic. In the audit-only red-team harness, post-capture mutation of the committed attention output is caught by the Merkle / bridge checks, but a **consistent** fake-a substitution, where the adversary chooses a fake post-attention output and re-derives the downstream trace honestly, passes the audit-only verifier. In the initial A1 toy sweep, all 10/10 consistent fake-a openings verified, and 4/10 also changed the emitted answer. This does not model the full production profile, which additionally audits witnessed scores, KV provenance, GQA / RoPE / mask wiring, and captured-logits decode; however, none of those checks re-derive arbitrary-position $\text{softmax}\left(Q\frac{K^T}{\sqrt{d}}\right)V$, so the residual class remains in scope until attention itself is verified or the audit bracket is strengthened.

Routine KV provenance is correspondingly weakest against sparse tampering: if an adversary corrupts only a few prefix positions, the per-audit detection probability under small-k sampling can be low. The short audit window also creates a denial-of-audit surface if a provider can force responses past the retention horizon before a challenge arrives. In practice, these risks push deployments toward automated audits, explicit response-deadline policies, longer or durable retention for high-value responses, and deeper audit modes when sparse prefix manipulation is a concern.

Several extensions could tighten the guarantees:

Per-step Q retention (extending the score-anchor scope). Under the current witness contract the prover retains Q for the final decode step only, so the score-anchor and causal-mask sub-audits are scoped to the last generated token. Retaining per-step Q would let the verifier audit pre-softmax scores at arbitrary generated positions without a sidecar redesign, at additional retained-state cost. This is the most direct way to widen the audit-only stock-mode attention claim while keeping the protocol architecture unchanged.

Output-conditioning analysis. The shell already verifies W_o exactly, but the current paper does not yet quantify how much adversarial freedom can remain after an unverified attention perturbation flows through W_o and the residual stream into the next opened layer. A layer-wise conditioning analysis would translate the audit-only bracket into an explicit downstream output bound.

Stronger KV provenance. The verifier can check all prefix positions simultaneously using random linear combinations: pick random scalars $\alpha_1 ... \alpha_n$ and check $v \cdot (\sum \alpha_i x_i) \stackrel{?}{=} r^T \cdot (\sum \alpha_i z_i)$. One check covers all n positions with false-accept probability $\leq 1/p$ (about $1/(2^{32}-5)$ in the current field). This upgrades KV provenance from statistical sampling to exact weight verification at all positions. Cost: $\sim 4\times$ audit bandwidth, making it better suited as an optional deep audit mode.

Broader empirical matrix. The current corridor measurements (reported as background in Section 6) cover Qwen2.5-7B-W8A8 and Llama-3.1-8B-W8A8. Immediate next validation targets are one additional family (e.g., Mistral or Gemma) and one materially larger model, so the audit-only attention story and routine audit policy are supported beyond the first two families.

Formalization and broader coverage. Ongoing follow-on work also includes Lean formalization of the core verification claims, validation on additional model families and larger checkpoints, stronger KV provenance, and W_o conditioning to translate the audit-only attention bracket into tighter downstream output bounds.

10 Conclusion

CommitLLM demonstrates that meaningful verification of LLM inference is possible without modifying the serving path or requiring expensive proof systems. The protocol's honest decomposition into exact, audited-but-not-verified, statistical, and fail-closed layers allows clients to understand precisely what is and is not guaranteed. Sampled decode is exact via captured GPU logits plus an algebraic LM-head binding; arbitrary-position attention outputs on stock GPU kernels are explicitly **not** verified, and stock-mode attention is bracketed by audits on its inputs and wiring (score anchoring, KV provenance, GQA / RoPE-config / causal-mask). Under the current witness contract the score-anchor and causal-mask sub-audits are scoped to the last generated token. The measured online serving overhead from tracing is currently about 12–14% across the tested 7B/8B/70B runs; heavier receipt finalization remains off the critical user-facing path, and on Llama 70B verifier cost is about 1.3 ms per challenged token under the measured 10-layer routine audit, rising to about 10 ms for a 1-token full audit over all 80 layers on a commodity CPU.

The immediate next steps are to extend the score-witness contract to per-step Q retention (widening the score-anchor and causal-mask scope beyond the last generated token), continue the Lean formalization, quantify downstream freedom through W_o conditioning, strengthen KV provenance, reduce post-generation finalization overhead, and extend the empirical matrix to additional families and larger models. These are now incremental improvements on top of a functioning protocol rather than rescue work for a broken design.

References

[1] Llama Team, AI @ Meta, "The Llama 3 Herd of Models." [Online]. Available: https://arxiv.org/abs/2407.21783

[2] Z. Ghodsi, T. Gu, and S. Garg, "SafetyNets: Verifiable Execution of Deep Neural Networks on an Untrusted Cloud," in *Advances in Neural Information Processing Systems 30*, 2017. [Online]. Available: https://papers.nips.cc/paper/7053-safetynets-verifiable-execution-of-deep-neural-networks-on-an-untrusted-cloud

[3] T. Liu, X. Xie, and Y. Zhang, "zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2968–2985. doi: 10.1145/3460120.3485379.

[4] H. Birkholz, D. Thaler, M. Richardson, N. Smith, and W. Pan, "Remote ATtestation procedureS (RATS) Architec-

ture," technical report RFC9334, Jan. 2023. doi: 10.17487/RFC9334.

[5] R. Freivalds, "Fast Probabilistic Algorithms," in *Mathematical Foundations of Computer Science 1979*, in Lecture Notes in Computer Science, vol. 74. Springer, 1979, pp. 57–69. doi: 10.1007/3-540-09526-8_5.

[6] R. C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," in *Advances in Cryptology – CRYPTO '87*, in Lecture Notes in Computer Science, vol. 293. Springer, 1987, pp. 369–378. doi: 10.1007/3-540-48184-2_32.

[7] W. Kwon *et al.*, "Efficient Memory Management for Large Language Model Serving with PagedAttention," in *Proceedings of the 29th ACM Symposium on Operating Systems Principles*, 2023, pp. 611–626. doi: 10.1145/3600006.3613165.

[8] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness," in *Advances in Neural Information Processing Systems 35*, 2022. [Online]. Available: https://arxiv.org/abs/2205.14135

[9] S. Goldwasser, S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems," *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–208, 1989, doi: 10.1137/0218012.

[10] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating Computation: Interactive Proofs for Muggles," in *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, 2008, pp. 113–122. doi: 10.1145/1374376.1374396.

[11] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A Cryptographic Inference Service for Neural Networks," in *29th USENIX Security Symposium (USENIX Security 20)*, USENIX Association, 2020, pp. 2505–2522. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/mishra

[12] J. Ménétrey *et al.*, "Attestation Mechanisms for Trusted Execution Environments Demystified," in *Distributed Applications and Interoperable Systems*, in Lecture Notes in Computer Science, vol. 13272. Springer, 2022, pp. 95–113. doi: 10.1007/978-3-031-16092-9_7.