Hollow-LLM Attack: Computationally Trivial Weights in Zero-Knowledge Verification of LLM Inference

Chen Gong
University of Southern California
cgong459@usc.edu

Beijie Liu
University of Southern California
beijieli@usc.edu

Mengyuan Li
University of Southern California
mli49061@usc.edu

***Abstract*—As large language models (LLMs) grow in scale and are predominantly served from remote platforms, verifying faithful inference execution becomes critical (i.e., ensuring that a provider actually executes the advertised model and computational workload rather than a tampered or downsized variant). Zero-knowledge (ZK) LLM inference offers an appealing approach. It promises public verifiability and delivers per-instance guarantees of equational correctness by proving that an output is consistent with executing a public architecture under committed, private weights. Though, we show that it does not bind the effort expended to produce the output.**

In this paper, we formalize this overlooked effort gap and introduce the Hollow-LLM Attack, in which a dishonest provider retains the declared architecture and parameter count but embeds ghost weights whose algebraic structure collapses effective computation. These witnesses satisfy the verification circuit and yield valid proofs, even though the dishonest model owner, who serves as the prover, performs computation commensurate with a much smaller model than the declared public architecture. This creates a profitable equilibrium in which providers deliver provably correct outputs at small-model cost while overclaiming model size. Accordingly, we characterize concrete families of ghost weights that compose with standard transformer blocks and show that such hollow deployments substantially reduce serving cost with zero quality loss under the same verification circuit. These findings underscore that proof of correct inference is not proof of large-model execution and necessitate additional protections to bind correctness to verifiable computational work.

1. Introduction

Large Language Models (LLMs) have become increasingly powerful and pervasive, underpinning a large market of AI services. With the growth in both size and scale, LLMs usually require substantial GPU resources and are thus often deployed on remote or cloud platforms. This deployment model raises critical concerns about the *faithful execution* of inference: *how can users trust that an LLM service is faithfully running the claimed model?* Such concerns include verifying that the deployed model has not been tampered with (e.g., no hidden backdoors) [1], [2], [3], that the outputs are truly generated by the advertised model [4], [5], [6], [7], [8], [9], [10], and that the model's size or parameters have not been surreptitiously replaced with a smaller or different model [11], [12], [13], [14], [15], [16].

To address this trust issue, the community has explored various privacy-preserving verification techniques. Approaches such as confidential computing (trusted execution environments) [17], [18], [19], homomorphic encryption [20], [21], [22], [23], [24], and zero-knowledge proofs (ZKPs) [5], [6], [7], [9], [10], have been proposed to enforce that an LLM inference was performed faithfully by the intended model without revealing proprietary model weights. Among these, zero-knowledge machine learning (zkML) stands out as a promising direction for publicly verifiable inference. A typical zkML workflow for LLM inference proceeds as follows: (i) the provider makes public the model architecture and a *commitment* to the proprietary private model weights (e.g., via a cryptographic hash); (ii) the user sends a prompt to the provider; (iii) the prover (i.e., the provider) generates a succinct cryptographic proof asserting that the produced output results from executing the declared model on the given prompt; and (iv) the verifier (i.e., the user) checks this proof. Crucially, the proof hides the actual weight values while guaranteeing that the output is consistent with committed private weights under the declared architecture. This commitment-only design is especially appealing for proprietary ML services because it offers public verifiability without revealing weights or relying on a trusted third party.

Recent systems demonstrate the feasibility of end-to-end ZK LLM inference even for multi-billion-parameter models, yielding compact proofs in practical time. For example, zkLLM [6] generates a proof for a 13B-parameter transformer in under 15 minutes with proof size around 200 KB, preserving model-weight privacy. Similarly, zkGPT [7] proves GPT-2 inference in about 25 seconds by introducing optimized constraints for transformer layers. These advances show that ZKP can ensure users of correct inference by large language models, instilling confidence that the model output was indeed correct inference results from the declared model.

In this paper, we reveal a previously overlooked limitation of the above ZK LLM inference verification procedure. Specifically, a zero-knowledge proof of inference certifies membership in a nondeterministic polynomial-time (NP) relation: it proves that there exist private weights

such that the output is the result of executing the public model architecture on the input. However, the proof does not attest to the algorithmic path taken to obtain that result or how much computation was necessary to get the result. In other words, it verifies the correctness of the equations that define the model size and architecture, but not the effort expended to calculate the result. This creates an **effort gap** compared to the declared public model size: *a dishonest provider can furnish a trivially valid witness that yields a verifiable output while performing far less work than the declared public architecture suggests.* We emphasize that this issue is not a flaw in ZKP soundness. Rather, it is a deployment-level assurance gap: the proof certifies consistency with a committed witness under the declared computation, but that guarantee does not inherently bind the amount of computation actually expended.

We concretize this gap through the *Hollow-LLM attack*. Our setting follows existing zero-knowledge LLM-inference deployments, such as zkGPT [7] and zkLLM [6], where the architecture is declared public, the weights are private but cryptographically committed, and the proofs accompany inference or periodic audits. Within this setting, a dishonest provider can deploy a *hollow LLM*: the provider publicly declares a larger model (the ***Outer Model***), then carefully fills the larger architecture with a smaller model augmented by *ghost weights*, and finally commits the entire hollow model. These ghost weights are chosen so that the declared layers either perform negligible computation or reuse the computation of the much smaller ***Inner Model***. During service, the provider computes outputs using only the smaller inner model. However, because the ghost weights are crafted appropriately, the provider can still supply a proof asserting that the output was computed by the declared larger outer model, under the committed weights and on the same input prompt. In this hollow LLM deployment, all public interfaces remain intact. The layer layout, tensor shapes, and stated parameter count all match the declared model, so both the proof and associated metadata appear legitimate. The deviation lies entirely in the private algebraic structure of the ghost weights: some layers effectively pass their inputs through unchanged, while wide layers are arranged so that only a small subspace carries the actual signal and the remainder remains inactive. Consequently, the computation follows that of the smaller inner model while the verifier continues to endorse the claimed outer model. Formal definitions and concrete constructions appear in Section 3 and Section 4.

We consider the Hollow-LLM attack particularly concerning for several reasons: **(1) Incentive to exaggerate the scale.** In the release and service of the LLM model, the number of parameters remains the most visible indicator of the capability and is continuously increasing; and companies prominently advertise the size of the model during public releases (the trend is illustrated in Figure 1). Previous studies also show that users tend to associate larger models and longer results with greater precision and reliability [25], [26], [27], further motivating providers to overstate the scale of their models. **(2) Resource-linked incentives.** Many pricing and governance schemes tie resource quotas and costs directly to the declared model size. Inflating the nominal parameter count can, therefore, yield both reputational and financial advantages. **(3) Proof-level indistinguishability** At the proof level, Hollow-LLM is covert: accepted transcripts remain consistent with the declared architecture and committed weights. Neither party can distinguish a hollowed model from a genuine one by comparing zero-knowledge proofs alone, since both produce proofs consistent with the public architecture. Existing clients and verifiers cannot *reliably* and *directly* infer the true model size because the model performance does not scale linearly with the parameter count [12], [15], [28], [29]. Deployment-level audits such as behavioral checks, reference-based comparisons, or telemetry may still detect aggressive over-claiming, but evaluating such detection signals is beyond the scope of this paper. Our focus here is the proof-level gap. **(4) Cost reduction.** A dishonest provider can generate outputs that satisfy the ZK verification circuit and produce valid proofs while performing only a fraction of the computation implied by the declared model.

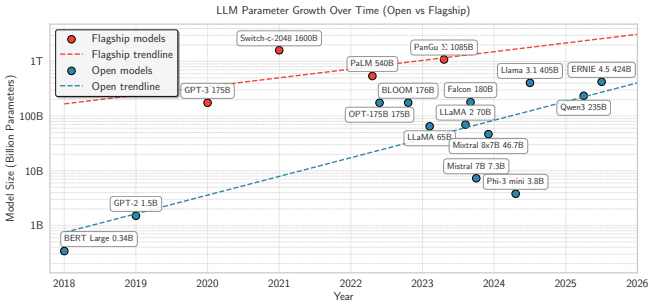

Figure 1. Growth trend of LLM parameters over time, distinguishing open-source and flagship models.

Notably, we observe that this kind of misdirection appears to be underappreciated in the emerging zkML community. several major blockchain and crypto infrastructure sources present ZK LLM inference in ways that implicitly equate proof generation with computational effort. For example, the Ledger Academy glossary, maintained by Ledger, one of the largest global crypto hardware wallet manufacturers, states that a ZK LLM inference "certificate contains details such as model size and parameters, confirming that a certain computation has been done" [30]. Likewise, CoinGecko, a leading cryptocurrency market data provider, defines ZK LLM as a system that verifies inference "including details like model size and parameters" [31]. These descriptions reveal a widespread misconception that ZK proofs of LLM inferences inherently attest to computational scale, highlighting why the Hollow-LLM attack and the resulting effort gap constitute a pressing and emerging concern.

The contributions of this paper are summarized as:

- We identify and formalize an *effort gap* in current ZK-verified LLM inference: existing schemes certify membership in an NP relation but provide no guarantee that the prover expends computation commensurate with the advertised model size, exposing a mismatch between what

ZK proofs attest to and the assurances assumed in LLM service markets.

- We introduce the Hollow-LLM attack model, in which a dishonest provider publicly declares a large outer model while privately deploying a smaller inner model. Using *ghost weights*, the provider preserves the public architecture, parameter count, and proof validity while routing almost all computation through the inner model, leaving verifiers unable to distinguish a hollowed deployment from a genuine one.
- We develop two training-free algebraic constructions of hollow LLMs that are compatible with state-of-the-art zkLLM pipelines. These constructions exploit transformer invariances to insert effectively identity layers and to inflate embedding dimensions while confining activations to a low-dimensional subspace, thereby decoupling publicly claimed depth and width from the true per-token computational cost.
- We analyze the security and economic implications of the effort gap in realistic deployment scenarios, showing that a provider can pass ZK verification while using only a fraction of the computation implied by the declared model. This creates strong incentives to exaggerate model scale and challenges emerging zkML narratives that equate proof validity with faithful execution effort.

2. Background

2.1. Transformer Architecture

In this paper, we focus on the standard decoder-only Transformer architecture used in contemporary LLMs and inference serving. Figure 3 illustrates a single pre-LN Transformer block, and Table 1 summarizes the notation for both the base architecture and the additional symbols introduced later for Hollow-LLM attacks. Here we briefly fix conventions for a single layer.

Let $x_{1:T}$ denote an input token sequence of length T, and let $X \in \mathbb{R}^{T \times d_{\text{model}}}$ be the corresponding stack of hidden states after token embedding and positional encoding. We adopt the usual factorization $d_{\text{model}} = h\, d_k = h\, d_v$ across h attention heads (see Table 1), and write the layer computation in pre-LN form as

$$\tilde{X} = \mathrm{LN}(X), \quad Q = \tilde{X} W_Q,\ K = \tilde{X} W_K,\ V_{\text{att}} = \tilde{X} W_V,$$
$$H_{\text{att}} = \mathrm{softmax}\Big(\tfrac{1}{\sqrt{d_k}} Q K^\top\Big) V_{\text{att}}, \quad X^{\text{att}} = X + H_{\text{att}} W_O,$$
$$\tilde{X}' = \mathrm{LN}(X^{\text{att}}), \quad X' = X^{\text{att}} + \varphi(\tilde{X}' W_1)\, W_2,$$

followed by a final layer normalization $\mathrm{LN}_{\text{final}}$. Here LN denotes a learned LayerNorm applied row-wise, W_Q, W_K, W_V, W_O are the attention projection matrices, W_1, W_2 are the feedforward weights, and φ is the pointwise nonlinearity (e.g., GELU).

The input pipeline maps discrete tokens in the vocabulary to rows of X via an embedding matrix E together with a positional mechanism P. The output pipeline maps the final hidden state to logits via a decoder matrix D, which may be tied or untied with E depending on the deployment.

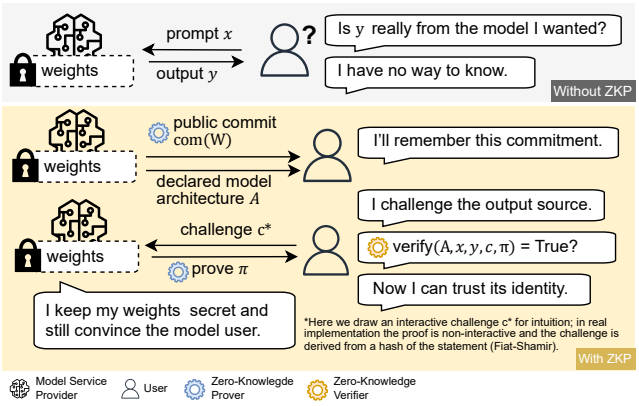

Figure 2. Ordinary serving versus zero-knowledge serving. The provider commits to private weights W and returns (y, π); the verifier checks that the declared architecture A, with some opening of $\mathsf{Com}(W)$, produces y on input x without revealing W.

2.2. Zero-knowledge LLM inference

Figure 2 contrasts ordinary remote LLM serving with a zero-knowledge deployment [6], [7]. A user sends a prompt x to a service provider and receives an output y. The model architecture A (layer layout, widths, heads, positional encoding, normalization scheme, and tying choices) is public, while the weights W remain private. The provider publishes a binding commitment $\mathsf{Com}(W)$ to these weights and, for each query, returns both the output y and a proof π.

In this setting, the provider acts as the *prover*, while the user or auditor acts as the *verifier*. We use *inference execution* to mean the actual forward-pass computation performed by the provider on input x. We say the service performs *faithful inference* when the returned output is exactly the result of running the declared architecture A with the committed private weights W on the same input. The public *statement* consists of $(A, x, y, \mathsf{Com}(W))$, while the private *witness* consists of the opening of the commitment and, depending on the proving system, any auxiliary values needed to satisfy the circuit.

The verifier runs

$$\mathsf{Verify}\big(A, x, y, \mathsf{Com}(W), \pi\big) \in \{\mathsf{accept}, \mathsf{reject}\}$$

and accepts exactly when there exist weights W that open the commitment and make the declared architecture A produce y on input x. In practice, an underlying interactive protocol is made non-interactive via the standard Fiat–Shamir transform, so the verifier simply checks a single proof π for the public statement $(A, x, y, \mathsf{Com}(W))$.

Overall, the proof certifies that the served output is *consistent* with some fixed private weights under the public architecture, without revealing those weights. This guarantee is commitment-relative: it ties the output to the committed private weights under the public architecture, rather than to any external reference model or public weight hash. Equivalently, acceptance establishes equation-level correctness of the claimed inference relative to the declared statement,

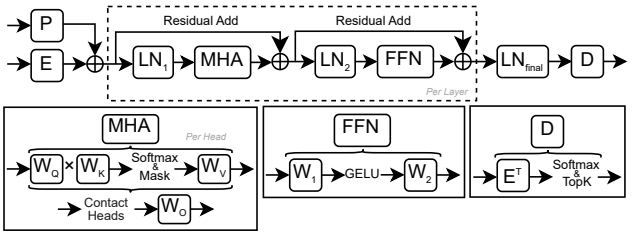

Figure 3. Orientation diagram for the pre-LN transformer: embedding E, positional P, attention (W_Q, W_K, W_V, W_O), feed-forward (W_1, W_2), and decoder D. Components outside the proof boundary are system dependent and stated when relevant.

TABLE 1. **SYMBOLS & NOTATION.** CORE TRANSFORMER SYMBOLS AND ATTACK-SPECIFIC SYMBOLS.

GROUP I: SYMBOLS USED IN TRANSFORMER

Symbol	Meaning
$x_{1:T}$, T	input token sequence and its length
y (or $y_{1:K}$)	next-token output (or K-step decode)
V	vocabulary size
d_{model}, h, d_k, d_v, d_{ff}, L	model width, heads, per-head dims, FFN width, depth
E, D	token embedding and decoder (tied unless noted)
P	positional encoding (absolute or rotary)
W_Q, W_K, W_V, W_O	attention projection matrices
W_1, W_2; φ	FFN matrices; φ nonlinearity
$\text{LN}_{\gamma,\beta}$	LayerNorm with $\gamma, \beta \in \mathbb{R}^{d_{\text{model}}}$

GROUP II: SYMBOLS USED IN THE ATTACK

Symbol	Meaning
A_{out}	declared public architecture checked by the verifier
A_{in}	served inner architecture executed by the prover
$W_{\text{out}}^{\text{ghost}}$	ghost witness weights under A_{out}
W_{in}	private weights for A_{in}
m	replication factor (Attack B)
d'_{model}	expanded width $m\, d_{\text{model}}$
$R = \mathbf{1}_m \otimes I_{d_{\text{model}}}$	replication from d_{model} to $m d_{\text{model}}$
S_s	selector for block $s \in [m]$
$\text{blkdiag}(M, \ldots, M)$	block-diagonal with m copies of M

but does not by itself certify how much computation or execution effort was expended to produce that output.

2.3. Easy Witnesses in LLM Serving

Work on verifiable computation has long noted the possibility of "easy witnesses": witnesses that satisfy all circuit constraints while corresponding to much less computation than a naive effort model would suggest [32], [33], [34]. In zk-LLM serving, this appears as a deployment-level effort-binding gap: the proof certifies consistency with the declared circuit and committed weights, but does not inherently bind the prover to the amount of computation users may associate with the declared model size.

To our knowledge, prior work has not concretely instantiated this gap for transformer-based zk-LLM serving. We show that transformer architectures admit structured easy witnesses, realized as *ghost weights*, that preserve the public architecture and proof validity while collapsing serve-time computation to that of a much smaller inner model.

3. Hollow-LLM Overview

A Hollow-LLM attack is a non-faithful LLM inference serving deployment in which a provider serves queries using a smaller inner model A_{in}, while supplying carefully chosen ghost weights so that the verifier accepts the transcript as if it were produced by the declared outer architecture A_{out} under the zk-LLM proof contract. In this section, we formalize the threat model, the attack objectives, and the target LLM architecture used in the hollow LLM deployment.

3.1. Threat Model

We consider proprietary-weight zkLLM inference deployments, consistent with zkLLM [6] and zkGPT [7]. The declared architecture A_{out} (layer topology, widths, heads, positional encoding, normalization layout, and any parameter tying) is public and compiled into the verifier's circuit. The model owner keeps weights (W_{out}) private and either commits to them in advance or supplies openings consistent with a prior commitment. We refer to the declared pair $(A_{\text{out}}, W_{\text{out}})$ as the ***Outer model***. Given an input prompt x, the provider returns an output y and a zero-knowledge proof π that the circuit's forward pass on $(A_{\text{out}}, W_{\text{out}}, x)$ yields y, revealing nothing about the weights. Public-weight or reference-bound deployments are therefore out of scope.

Arithmetic and Determinism. The circuit implements a deterministic transformer forward pass using fixed-point/integer arithmetic with public rounding rules; nonlinearities (e.g., softmax, GELU) are realized via public lookup tables. Stochastic decoding, if used, is made reproducible by a public, committed seed that fixes the sampling trajectory. This ensures that the same (x, W) and seed deterministically yield the same y. Quantization is a common implementation choice in most verifiable ML to match verifier arithmetic and avoid floating-point nondeterminism; the hollowing constructions themselves are algebraic and apply equally to floating-point models so long as prover and circuit share the same numeric semantics.

Inference Serving vs. Proof Regime. Inference is served token-by-token, whereas proofs are generated per query transcript. In practice, inference results may be returned without real-time proof for non-interactive ZK proofs ([7]): when a specific call is selected for audit, the provider can reconstruct a witness for the already-served transcript (by re-running the deterministic computation or using cached intermediates) and produces a proof π off the critical path. This regime avoids the overhead of attaching a ZK proof

to every response, while still enabling post-hoc audits that deter blatant cheating.

Adversary Capabilities. The adversary is a dishonest service provider who seeks to reduce computation while appearing to run the larger declared model. Concretely, the provider may generate y using a smaller ***Inner Model*** $(A_{\text{in}}, W_{\text{in}})$ at serve time, while convincing the verifier that y came from A_{out}. The attacker selects *ghost weights* $(W_{\text{out}}^{\text{ghost}})$ so that the circuit accepts (x, y) as consistent with A_{out}, while y can in fact be produced at the cost of executing only the smaller model A_{in}. Typically A_{out} strictly expands A_{in} (e.g., deeper or wider), enabling algebraic alignment between the outer equations and a cheaper computation path. The provider controls the online service and also acts as the prover, including batching, caching, scheduling, and persistence of intermediate tensors across serve, generating proof, as well as participating during the audit phases. The commitment is binding across serving and audit; the attacker's freedom lies in choosing the committed weights within this private-model regime, not in changing them after the fact. The adversary cannot modify the verifier's circuit, trapdoors, or keys, and learns nothing beyond what the zero-knowledge procedure reveals.

Verifier View. For each verification, the verifier (or the service client) observes only (x, y, π) and an accept/reject bit. There is no trusted hardware attestation bound to runtime behavior, no external metering of FLOPs/energy, no side-channel telemetry, and no watermarking or behavioral fingerprinting assumed. Audits may be sparse and asynchronous relative to serving, reflecting practical deployments where proving is performed on demand or periodically rather than on every token. Verification runs the fixed circuit for A_{out} and is lightweight relative to proving (Section 5).

Out of scope. We exclude training provenance and dataset integrity, key management, network-layer timing attacks, and defenses that alter the circuit to measure work or introduce orthogonal trust anchors (e.g., TEEs with attestation). Our focus is the gap between *equation-level correctness* and *expended effort* in conventional ZK LLM workflows. The Hollow-LLM deployments instantiated later exploit these degrees of freedom and the audit-based operation to shift cost off the critical path, aiming at the goals in Section 3.2.

3.2. Attack Objectives

The main objective of the Hollow-LLM attack is to make a dishonest provider appear fully compliant with the verifier, while the provider can pay only the computational cost of a smaller model during inferencing. To match practical hollow LLM deployments, we target five concrete attack objectives:

- ***G1.*** *Contract preservation.* We do not modify the verifier, its circuit, the statement format, or public parameters (CRS, keys, API). Each proof still asserts that y equals the forward pass of the declared architecture on x.
- ***G2.*** *Asymmetric serving and proving.* Online inference generation uses a smaller inner model $(A_{\text{in}}, W_{\text{in}})$, while the audit-time proof is issued against A_{out} using private *ghost weights* $W_{\text{out}}^{\text{ghost}}$.
- ***G3.*** *Serve-time efficiency.* Per-token latency and FLOPs at serve time should match A_{in}, not A_{out}; in practice, the provider can add artificial delay, so raw response time alone is not a reliable distinguisher.
- ***G4.*** *Proof indistinguishability.* For fixed (A_{out}, x, y) and a fixed circuit, accepted proofs from hollowed runs are distributed the same way as those from an honest run of A_{out}.
- ***G5.*** *Cost asymmetry.* Any extra effort is shifted off the latency path to the audit/proving phase (one proof per transcript), while the verification cost remains small as in standard zkML.

3.3. General Attack Steps

We outline the general Hollow-LLM attack steps below:

1) Declared Public Architecture and Committed Private Weights. The model provider originally has an inner model $(A_{\text{in}}, W_{\text{in}})$. The provider then privately chooses an appropriate size A_{out} and attack-specific *ghost weights* $W_{\text{out}}^{\text{ghost}}$.The provider then declares the model architecture and commits W_{out}. With prompt input (x), a set of committed weights, and a public fixed or derived decoding seed s, the later LLM inference serving service can be verified by ZKP.
2) LLM Serving (per token, fast). The provider runs the inner model and returns tokens to the user, yielding
$$y_{\text{in}} \;=\; f_{A_{\text{in}}}(W_{\text{in}}, x; s).$$
This phase executes only A_{in} at serve time. Wall-clock latency may be padded, so raw response time alone is not a definitive distinguisher.
3) ZKP Audit (per transcript, off the critical path). The user (i.e., verifier) can periodically choose a pair (x, y_{in}) and ask the model provider for a ZK proof that the previous response was generated by $(A_{\text{out}}, W_{\text{out}})$. The attack-specific ghost weights $W_{\text{out}}^{\text{ghost}}$ must be carefully crafted in STEP 1 for the declared architecture A_{out} so that evaluating the verifier's fixed circuit on the same (x, s) reproduces the already served transcript. This should not be a one-shot coincidence, but should hold for any (x, s) pair, since any change in W_{out} would be reflected in the commitment and detected by the user :
$$f_{A_{\text{out}}}(W_{\text{out}}^{\text{ghost}}, x; s) \;=\; y_{\text{in}}.$$
4) Between Serving and Audit (optional caching). The provider may optionally cache minimal state (e.g., the public seed, prompts, or selected intermediates) to speed up the auditing procedure. This is not required and is not visible to the verifier.
5) Produce the Proof. Per audit request, the provider generates a proof π using standard zkML machinery for the *outer* circuit on (x, y_{in}), and the verifier checks it exactly as in an honest run.

Success Criteria. We regard the attack as successful only if it simultaneously achieves correctness and the desired cost asymmetry. We focus on per-serving cost, since serving happens token by token and determines the model provider's cost of the service, whereas proving happens only on demand: the cost of generating a ZKP for LLM inference is order-of-magnitude times that of inference serving and can be scheduled asynchronously (21.8s for GPT-2 using 32 threads [7]). By construction,

$$T^{\text{serve}}_{\text{hollow}} \;=\; T^{\text{serve}}(A_{\text{in}}) \quad \ll \quad T^{\text{serve}}(A_{\text{out}}),$$

so the provider saves serve-time cost. On an audited call, we regard the attack as successful when all criteria during verification hold:

- ***S1.*** *Accepted Proof.* The verifier accepts a ZK proof for the declared circuit on the served transcript: $\text{Verify}(A_{\text{out}}, x, y_{\text{in}}, \pi) = \text{accept}$.
- ***S2.*** *Exact Transcript Alignment.* The served output is exactly reproduced by the declared architecture under ghost weights on the same input and seed: $y_{\text{in}} = f_{A_{\text{out}}}(W^{\text{ghost}}_{\text{out}}, x; s)$.
- ***S3.*** *Indistinguishability.* For fixed (A_{out}, x, y) and circuit, accepted proofs from hollowed runs are (up to negligible differences) distributed the same way as those from honest runs.

4. Hollow-LLM Attack

Building on the proof contract and notation in Section 3, we now instantiate Hollow-LLM attack with two training-free ghost-weight constructions, *Attack A* and *Attack B*. Both exploit algebraic invariances of pre-LN Transformer blocks to embed a smaller served *inner* model A_{in} into a larger, verifiable *outer* architecture A_{out} while preserving the input–output mapping.

4.1. Algebraic Invariances of Transformer Blocks

Naïvely, it is *not* obvious that one can hollow out a Transformer simply by adding layers or replicating coordinates. Residual connections, LayerNorm, nonlinearities, and multi-head attention tightly couple coordinates, so arbitrary depth or width inflation usually changes the logits (*inner model and committed outer model then have **different** output*), even if the new parameters look "small" or "unused." However, we identified that they nonetheless admit a small set of algebraic invariances that allow rigorously controlled structural manipulation. These invariances characterize exactly when added depth behaves as a structural identity and when width expansion preserves the input–output function. They are the algebraic backbone that makes both Attack A and Attack B possible. In what follows, we isolate four such invariances. Each appears simple in isolation, but together they determine when a pre-LN Transformer block can be expanded along depth or width without altering its functional behavior. We later use these properties to construct zero-computation layers (Attack A) and width-replicated blocks (Attack B). We adopt the pre-LN architecture from Section 2.1. For a single attention block followed by an FFN block, the pre-LN update is

$$x'_\ell = x_\ell + F_{\text{attn},\ell}\big(\text{LN}_{\gamma^{\text{attn}}_\ell, \beta^{\text{attn}}_\ell}(x_\ell)\big),$$
$$x_{\ell+1} = x'_\ell + F_{\text{ffn},\ell}\big(\text{LN}_{\gamma^{\text{ffn}}_\ell, \beta^{\text{ffn}}_\ell}(x'_\ell)\big).$$

Here x'_ℓ denotes the intermediate representation after the attention sublayer. The key point is that each sublayer appears only inside a *residual add*.

Residual identity. Consider a generic pre-LN residual block

$$x \;\mapsto\; x + F(\text{LN}(x)).$$

If we choose parameters so that $F(\text{LN}(x)) = 0$ for all inputs in the support of the deployment (here, the quantized activations that can arise from the inner model under the verifier's arithmetic), then the block acts as an exact identity:

$$x \mapsto x + 0 = x.$$

A stack of such blocks contributes zero effective depth while still being fully compatible with the circuit. This property allows Attack A to insert arbitrarily many declared layers that contribute symbolic depth but perform no computation.

Elementwise nonlinearities. Let ϕ be any elementwise nonlinearity used in the FFN (e.g., GELU, ReLU, SiLU), realized via lookup tables in the circuit. For any vector z and replication factor $m \in \mathbb{N}$, define

$$R\,z \;\in\; \mathbb{R}^{md} \quad \text{by} \quad R\,z = (z, \ldots, z)$$

(the concatenation of m identical copies; we call R the *replication operator*). Then

$$\phi(0) = 0, \qquad \phi(R\,z) = R\,\phi(z)$$

because ϕ is applied coordinatewise. Thus zeros remain zeros under ϕ, and replicated coordinates remain exactly equal. This ensures that padding or replication does not introduce cross-coordinate interference and allows us to zero out padded FFN neurons in Attack A and to preserve width-wise replication in Attack B.

LayerNorm replicate invariance. For a hidden vector $x \in \mathbb{R}^d$, LayerNorm computes

$$\text{LN}_{\gamma,\beta}(x) = \gamma \odot \frac{x - \mu(x)\mathbf{1}}{\sigma(x)} + \beta,$$

where $\mu(x)$ and $\sigma(x)$ are the per-channel mean and standard deviation, and $\gamma, \beta \in \mathbb{R}^d$ are learned parameters. Now replicate x into $x' = R\,x \in \mathbb{R}^{md}$ using the same operator R as above. Since x' consists of m identical length-d blocks, its mean and variance match those of x:

$$\mu(x') = \mu(x), \qquad \sigma(x') = \sigma(x).$$

If we also replicate the LN parameters as

$$\gamma' = R\,\gamma, \qquad \beta' = R\,\beta,$$

then LayerNorm commutes with replication:

$$\text{LN}_{\gamma',\beta'}(x') = R\,\text{LN}_{\gamma,\beta}(x).$$

This *LayerNorm replicate invariance* is the core algebraic fact that makes Attack B possible: we can inflate the width by a factor m while preserving the normalized activations that drive attention and FFN computation.

Multi-head attention under replication. Write a single multi-head attention sublayer as

$$\mathrm{MHA}(x) = W_o\big(\mathrm{Concat}(h_1, \ldots, h_H)\big),$$

where each head h_j is computed from x via learned projections $W_q^{(j)}, W_k^{(j)}, W_v^{(j)}$ and a softmax. For Attack B, we construct an outer attention with width $d'_{\text{model}} = m\, d_{\text{model}}$ by:

- replicating the inner projections across m disjoint subspaces of the wider hidden state, forming a block-diagonal weight matrix, and
- replicating the hidden state into these subspaces using R.

Because the attention equations are linear in the projections and apply independently per head, each replicated block behaves as an exact copy of the inner attention. Combined with LayerNorm replicate invariance, this preserves the per-head score distributions and outputs exactly under the circuit arithmetic. These invariances are precisely the ingredients used below. Attack A uses residual identity and FFN zero-padding to inflate depth without changing the realized computation. Attack B uses replication together with block-diagonal attention/FFN structure to inflate width while keeping the logits unchanged.

FFN zero-padding. A two-layer FFN with hidden width d_{ff} computes

$$\mathrm{FFN}(x) = W_2\, \phi(W_1 x + b_1) + b_2.$$

If we enlarge d_{ff} by appending extra neurons whose incoming and outgoing weights are zero, then for any input we have

$$\phi(W_1' x + b_1') = \big(\phi(W_1 x + b_1),\, 0\big), \qquad W_2' = [W_2 \;\; 0],$$

so the FFN output remains unchanged. This lets Attack A conform to a larger declared FFN width without altering the computation; the added parameters are ghost parameters.

From invariances to attacks. We will next show that the combination of these invariance is sufficient to instantiate the two Hollow-LLM attacks described below. In Attack A (Section 4.2), we use the residual-identity property to insert arbitrarily many zero-computation layers that are visible to the verifier but invisible to the serve path. In Attack B (Section 4.3), we combine replication invariances, LayerNorm behavior, and block-diagonal attention/FFN structure to inflate the model dimension while preserving the logits.

4.2. Attack A: Hollowing Depth through Zero-Work Residual Blocks

Attack A increases the declared depth and, optionally, the FFN width of the model while keeping the effective computation depth L_{in} equal to that of the inner model A_{in}. The served model is embedded as a prefix of the declared

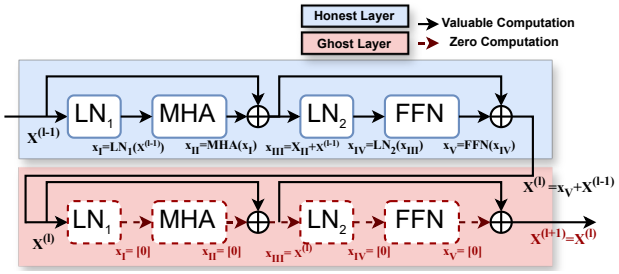

Figure 4. Attack A (add-zero-layers). The inner model's layers (solid) are embedded as a prefix of the outer model. The added layers (dashed) are configured so that both the attention and FFN sublayers output zero, leaving the residual path unaffected. Although the verifier observes a depth of L_{out} and an FFN width of $d_{\text{ff,out}}$, the effective computation still matches that of the depth-L_{in} inner model.

layers, and all remaining blocks are configured to behave as algebraic identities. These added layers are fully visible to the verifier but invisible to the serve path (Figure 4).

Recipe and subsection structure. Formally, we start from an inner architecture A_{in} with depth L_{in} and per-layer FFN width $d_{\text{ff,in}}$ and construct an outer architecture A_{out} with depth $L_{\text{out}} \geq L_{\text{in}}$ and FFN width $d_{\text{ff,out}} \geq d_{\text{ff,in}}$, matching A_{in} on all other public hyperparameters. Section 4.2.1 fixes this public shape and defines an expansion operator E_{layers} that copies the first L_{in} layers and fills the remaining layers with zero-computation residual blocks. Section 4.2.2 then shows how a prover who actually runs A_{in} can edit cached activations into a witness for A_{out}, so that the verifier's circuit reproduces exactly the inner-model transcript on the same (x, s).

Challenge and idea. Extra layers usually perturb LayerNorm statistics and residual scaling, altering logits even when their parameters are "small." Attack A uses the residual-identity and FFN zero-padding properties from Section 4.1 to set both the attention and FFN sublayers of added blocks to output exact zeros on the verifier's arithmetic domain, so each added block acts as $x \mapsto x$ and contributes depth only symbolically.

4.2.1. Public Shape and Ghost-weight Construction. The public outer model architecture describes:

- the larger depth L_{out},
- the larger FFN width $d_{\text{ff,out}}$,
- identical attention interfaces $(d_{\text{model}}, h, d_k, d_v)$, and
- identical embedding and decoder layers (E, D) and positional encoding.

Thus the verifier's circuit is locked to L_{out} blocks with FFN width $d_{\text{ff,out}}$ and will expect intermediate tensors of that shape in the witness.

Expansion operator $\mathcal{E}_{\text{layers}}$. We define a map

$$\mathcal{E}_{\text{layers}} : W_{\text{in}} \mapsto W_{\text{out}}^{\text{ghost}}$$

that copies the first L_{in} layers and fills the remaining layers with identities:

- For each layer $\ell \le L_{\text{in}}$, copy the attention and FFN parameters directly,

$$(W_q^\ell, W_k^\ell, W_v^\ell, W_o^\ell)^{\text{out}} = (W_q^\ell, W_k^\ell, W_v^\ell, W_o^\ell)^{\text{in}},$$
$$(W_1^\ell, W_2^\ell)^{\text{out}} = (W_1^\ell, W_2^\ell)^{\text{in}},$$

  with the same biases and LayerNorm parameters. If $d_{\text{ff,out}} > d_{\text{ff,in}}$, append rows/columns of zeros and zero biases to match the outer width.
- For each added layer $\ell > L_{\text{in}}$, set the attention and FFN sublayers to output zeros for all reachable inputs:

$$W_q^\ell = 0, \quad W_k^\ell = 0, \quad W_v^\ell = 0, \quad W_o^\ell = 0,$$
$$W_1^\ell = 0, \quad b_1^\ell = 0, \quad W_2^\ell = 0, \quad b_2^\ell = 0,$$

  with arbitrary (but fixed) LayerNorm parameters.

By residual identity and FFN zero-padding, each added block computes $x \mapsto x$ on the verifier's arithmetic domain.

Therefore, for every supported prompt and decoding seed, the outer model follows exactly the same hidden-state trajectory as the inner model through the first L_{in} blocks, after which only identity blocks remain. Equivalently,

$$f_{A_{\text{out}}, E_{\text{layers}}(W_{\text{in}})}(x; s) = f_{A_{\text{in}}, W_{\text{in}}}(x; s)$$

for all supported (x, s).

4.2.2. Witness Editing (serve → audit). At serve time, the dishonest provider runs only $(A_{\text{in}}, W_{\text{in}})$ and materializes activations only for the first L_{in} blocks. For each token position t, the provider computes the inner-model activations and logits and returns the next-token prediction y. Optionally, for anticipated audits the provider caches intermediate activations

$$\{x_{\ell,t}\}_{\ell=0}^{L_{\text{in}}}$$

for each token position in the transcript; these are internal and never exposed to the verifier.

Audit path. When an audit requires a zero-knowledge proof, the prover must supply a witness consistent with running the declared outer model $(A_{\text{out}}, W_{\text{out}}^{\text{ghost}})$. The witness editing procedure is:

1) For layers $\ell \le L_{\text{in}}$, copy all cached intermediate tensors (activations, attention scores, FFN outputs) from the serve-time run into the corresponding slots of the outer-model trace.
2) For FFN padded neurons (if $d_{\text{ff,out}} > d_{\text{ff,in}}$), set their pre-activations and post-activations to zero, which is consistent with the zero weights and $\phi(0) = 0$.
3) For added layers $\ell > L_{\text{in}}$, set the sublayer outputs to zero and the residual outputs equal to their inputs:

$$x_{\ell,t}^{\text{out}} = x_{\ell-1,t}^{\text{out}}, F_{\text{attn},\ell}(x_{\ell-1,t}^{\text{out}}) = 0, F_{\text{ffn},\ell}(x_{\ell,t}^{\prime\text{out}}) = 0.$$

Because the ghost weights were chosen so that the added blocks are algebraic identities, the verifier's circuit accepts this witness as a valid execution of A_{out}: for all $|x| \le L_{\text{in}}$,

$$f_{A_{\text{out}}, \mathcal{E}_{\text{layers}}(W_{\text{in}})}(x) = f_{A_{\text{in}}, W_{\text{in}}}(x).$$

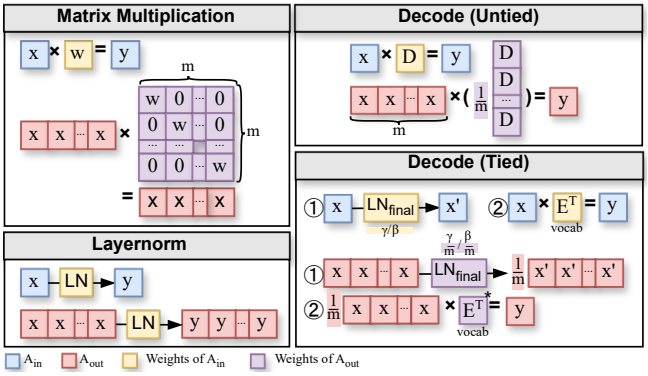

Figure 5. Attack B (enlarge-embedding by replication). The hidden state is replicated across m blocks of coordinates. Attention and FFN weights are arranged in block-diagonal and repeated patterns so that each block behaves like a copy of the inner model. The output layer is constructed so that logits match the inner model exactly (zero-padded or averaged decoder for untied, and a final $1/m$ rescaling step for tied).

Takeaway. Attack A increases the declared depth without changing the realized computation: the extra layers remain part of the verified outer circuit, but they contribute only residual identities during both serving and audit-time witness generation.

4.3. Attack B: Width Inflation via Replicated Coordinates

Attack B mirrors the same template along the width axis: it inflates the declared model dimension by replicating the hidden state across m coordinate blocks, arranges attention and FFN weights in block-diagonal / repeated form to forbid cross-block mixing, and then collapses the replicas at the output so that logits match those of A_{in} exactly (Figure 5).

Recipe and subsection structure. Let A_{in} be a pre-LN Transformer with model dimension d_{model} and weights W_{in}. We define an outer architecture A_{out} with expanded width $d'_{\text{model}} = m\, d_{\text{model}}$, the same depth L, the same dimension of heads, and the same context-length limit. Section 4.3.1 introduces the replication operator R and constructs ghost weights $E_{\text{embed}}(W_{\text{in}})$ layer by layer so that all hidden states in the outer circuit take the replicated form $x'_{\ell,t} = Rx_{\ell,t}$. Section 4.3.2 then shows that, given an inner run, simply replicating activations and adding a final $1/m$ rescaling step in the tied-decoder case yields a witness that satisfies every equation in the wider circuit and produces identical logits.

Challenge and idea. Naïvely widening the model changes dot-product scales and LayerNorm statistics, and tied decoders amplify these changes, so the softmax distribution is no longer preserved. Attack B combines the LayerNorm replicate invariance and per-block attention/FFN structure from Section 4.1 with an explicit output collapse (zero-padding or a final $1/m$ rescaling for tied decoders) to keep both logits and softmax outputs equal to those of the inner model.

4.3.1. Public Shape and Replicated Hidden State. At serve-time, the prover runs A_{in} and maintains hidden states $x_{\ell,t} \in \mathbb{R}^{d_{\text{model}}}$. At proof-time, the witness must present hidden states $x'_{\ell,t} \in \mathbb{R}^{d'_{\text{model}}}$ to the circuit of A_{out}. for every layer ℓ and every token position t in the served transcript,

i.e., the outer hidden state is always m concatenated copies of the inner hidden state. The LayerNorm replicate invariance from Section 4.1 ensures that normalized activations also satisfy this property.

Expansion operator $\mathcal{E}_{\text{embed}}$. We now describe how to construct the ghost weights layer by layer. Let $R = \mathbf{1}_m \otimes I_d$ as before, and let S_s select the s-th block of d coordinates from a length-md vector.

Embeddings and LayerNorm. For the token embedding and positional encoding:

- For each token v with inner embedding $E_v \in \mathbb{R}^d$, define
$$E'_v = RE_v \in \mathbb{R}^{md}.$$
-
- For any learned positional encodings p_t, set $p'_t = Rp_t$ for every supported token position t.
- For each LayerNorm with parameters (γ, β), set $(\gamma', \beta') = (R\gamma, R\beta)$.

By construction and the invariances of Section 4.1, all normalized inputs to attention and FFN are replicated.

Attention sublayers and softmax. For each layer ℓ and each head, let $W_q^\ell, W_k^\ell, W_v^\ell$ and W_o^ℓ denote the inner projections. We form the outer projections as block-diagonal replications:

$$W_q'^{\ell} = \text{blkdiag}(W_q^\ell, \ldots, W_q^\ell),$$
$$W_k'^{\ell} = \text{blkdiag}(W_k^\ell, \ldots, W_k^\ell),$$
$$W_v'^{\ell} = \text{blkdiag}(W_v^\ell, \ldots, W_v^\ell).$$

Given a replicated input $x' = Rx$, each block $S_s x'$ computes exactly the same queries, keys, and values as the inner head:

$$Q_s = W_q^\ell x, \quad K_s = W_k^\ell x, \quad V_s = W_v^\ell x \qquad \text{for all } s \in [m].$$

Crucially, the softmax in multi-head attention is taken over the *key positions*, not over the feature dimension. Replication only happens along the feature dimension: we do not create extra tokens or extra time steps. Therefore, for each head and each position, the score matrices

$$\frac{Q_s K_s^\top}{\sqrt{d_k}}$$

and the resulting attention weights $\text{softmax}(\cdot)$ are identical across all s and coincide with the inner model's attention weights. The concatenation of heads simply glues these replicated outputs along the feature axis.

For the output projection, we can either:

- *single-block routing:*
$$W_o'^{\ell} = W_o^\ell S_1^\top,$$
i.e., we apply W_o^ℓ to the first replicated block and ignore the others; or
- *averaging across blocks:*
$$W_o'^{\ell} = \frac{1}{m}\begin{bmatrix} W_o^\ell & W_o^\ell & \cdots & W_o^\ell \end{bmatrix},$$
which averages the m copies of the same head output.

In both cases, the attention sublayer output matches the inner attention output exactly for replicated inputs, and the softmax distributions over positions are unchanged.

FFN sublayers. For each FFN layer, we replicate both the input and hidden dimensions:

$$W_1'^{\ell} = \text{blkdiag}(W_1^\ell, \ldots, W_1^\ell),$$
$$W_2'^{\ell} = \begin{bmatrix} W_2^\ell & W_2^\ell & \ldots & W_2^\ell \end{bmatrix}.$$

Given a replicated input $x' = Rx$, we have

$$W_1'^{\ell} x' = R(W_1^\ell x), \qquad \phi(W_1'^{\ell} x' + b'_1) = R\phi(W_1^\ell x + b_1),$$

and then

$$W_2'^{\ell}\, \phi(W_1'^{\ell} x' + b'_1) = W_2^\ell\, \phi(W_1^\ell x + b_1),$$

so the FFN output exactly matches the inner FFN. Again, zeros and replication are preserved by the elementwise nonlinearity.

Output logits and softmax. Finally, we must ensure that the *logits* and therefore the softmax outputs seen by the verifier match those produced by the inner model. This is the place where naive "just multiply everything by m" reasoning breaks: changing the scale of logits will in general change the softmax distribution, and the circuit checks those exact arithmetic relations. We must therefore design A_{out} so that its logits are *exactly* the same as the inner logits on all supported transcripts.

We consider untied and tied decoders separately.

Untied decoder. If the output projection D is not tied to E, we have full freedom to choose D' without affecting embeddings. We maintain the replicated hidden state $h' = Rh$ and define D' so that $D'h' = Dh$ coordinatewise.

Two simple constructions satisfy this:

- *Zero-padded decoder:*
$$D' = \begin{bmatrix} D & 0 & 0 & \cdots & 0 \end{bmatrix},$$
i.e., D applied to the first block and zeros on the remaining $(m-1)$ blocks. Then
$$D'h' = Dh + 0 + \cdots + 0 = Dh.$$
- *Averaging decoder:*
$$D' = \frac{1}{m}\begin{bmatrix} D & D & \cdots & D \end{bmatrix}.$$
Since each block of h' is equal to h, we obtain
$$D'h' = \frac{1}{m}\sum_{s=1}^{m} Dh = Dh.$$

In both cases, the logits supplied to the softmax are *identical* to the inner logits. Therefore, the softmax outputs and

the next-token distribution are unchanged, and the verifier sees exactly the same equations as in an honest run of A_{in}.

Tied Decoder. When $D = E^\top$, the situation is more constrained. We still set $E'_v = RE_v$ so that embeddings respect replication, and we tie $D' = E'^\top$ as usual. For a replicated hidden state $h' = Rh$, the raw dot product logits satisfy

$$\langle E'_v, h' \rangle = \langle RE_v, Rh \rangle = m \langle E_v, h \rangle,$$

so naive tying introduces a uniform factor m on all logits. That factor *does* change the softmax (it sharpens the distribution) and breaks equality at the level of probabilities.

To fix this while preserving tying, we rescale the final hidden representation by $1/m$ *right before* applying the tied decoder. Concretely, let z_L be the pre-readout hidden state of the inner model and $z'_L = Rz_L$ its replicated counterpart. In the outer model we apply an extra scalar map αI with $\alpha = 1/m$ to the final LayerNorm output:

$$\tilde{z}'_L = \alpha\, z'_L = \frac{1}{m} R z_L.$$

We can fold this α into the scale parameter of the *last* LayerNorm or implement it as an explicit diagonal linear layer that is part of A_{out}. The logits then satisfy

$$\langle E'_v, \tilde{z}'_L \rangle = \left\langle RE_v, \frac{1}{m} R z_L \right\rangle = \frac{1}{m}\, m \langle E_v, z_L \rangle = \langle E_v, z_L \rangle,$$

so tying is preserved and the logits are exactly equal to those of the inner model.

It is important that this $1/m$ rescaling happens at the *last* LayerNorm (or immediately after the final residual block). If we attempted to scale at the very beginning of the network, pre-LN residual structure

$$x_{\ell+1} = x_\ell + F_\ell(\text{LN}(x_\ell))$$

would reintroduce unscaled contributions through the residual adds: different paths would carry differently scaled signals, and their sums would no longer match the inner representation. Only after the final residual add—when no further mixing occurs—can we safely apply a uniform scalar factor and preserve equality with the inner model.

4.3.2. Witness Editing and Correctness. Similar as Attack A, the prover actually runs A_{in} and caches the inner activations. To construct the witness, the prover:

1) replicates each inner hidden state $x_{\ell,t}$ into $x'_{\ell,t} = Rx_{\ell,t}$;
2) uses the ghost weights from $\mathcal{E}_{\text{embed}}(W_{\text{in}})$ for all sublayers; and
3) fills in the attention and FFN intermediates implied by the replicated computation, including the final $1/m$ rescaling step in the tied-decoder case.

By the invariances already established and the explicit treatment of the logits above, every layer equation in the outer circuit is satisfied, and both logits and softmax outputs match those of the inner model exactly.

Takeaway. Attack B increases the *reported* width by replicating coordinates and then collapsing them at the output, so the verifier sees a larger model but the logits and serve-path cost match A_{in}.

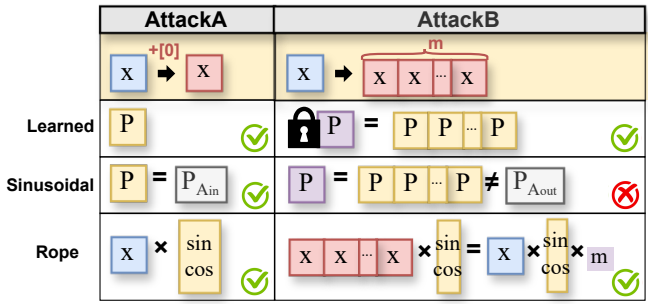

Figure 6. Positional encodings under both attacks. For learned encodings, both attacks can reuse or replicate the same table. For deterministic sinusoidal encodings, Attack A still matches the inner model, but Attack B cannot: the circuit recomputes P_{out} as a public function of $(t, d_{\text{model,out}})$, so in general $P_{\text{out}} \neq RP_{\text{in}}$ and the mismatch is ZK-detectable. For RoPE, we align rotation frequencies and replicate heads so that within-head indices match and rotations coincide.

4.4. Positional Encodings

Unlike most other components of the Transformer, whose basic blocks have remained fairly stable, positional encoding has evolved substantially over time. Early models used fixed sinusoidal functions; later variants moved to learned absolute tables; and modern LLMs increasingly adopt rotary encodings (RoPE) and related schemes. In this subsection we examine these major choices and analyze, for each, how compatible it is with our hollow LLM constructions.

Because the positional scheme is hard-coded in the verifier's circuit, both attacks must respect whatever encoding the outer architecture specifies. Figure 6 summarizes the three cases: learned encodings and RoPE remain compatible with both attacks after simple copying or replication, whereas deterministic sinusoidal encodings still support Attack A but can rule out Attack B.

Scope and Placement. We assume $|x| \leq L_{\text{in}}$, consistent with the inner model's context length. Under pre-LN, positional information is added before the first LayerNorm in each layer, so Attack A's zero blocks and Attack B's replication always act on states that already include positions. We only need the embedding + position step to respect the invariances in Section 4.1.

Learned absolute tables. For learned absolute embeddings $p_t \in \mathbb{R}^{d_{\text{model}}}$:

- *Attack A.* The outer model reuses the same table for $t \leq L_{\text{in}}$; added layers do not change p_t.
- *Attack B.* We define $p'_t = Rp_t$, so that $x'_0 = R(E(x_t)+p_t)$ and the replicate-invariance conditions hold.

Sinusoidal encodings. For deterministic sinusoidal encodings, the circuit computes $p_t = g(t, d_{\text{model}})$ as a fixed public function. Once $d_{\text{model,out}}$ is fixed, P_{out} is fully determined.

- *Attack A.* Since $d_{\text{model,out}} = d_{\text{model,in}}$, we have $p_t^{\text{out}} = p_t^{\text{in}}$ for $t \leq L_{\text{in}}$, and extra layers leave positions unchanged.
- *Attack B.* With $d_{\text{model,out}} = m\, d_{\text{model,in}}$ and $m > 1$, the circuit enforces $p'_t = g(t, d_{\text{model,out}})$, while we would need $p'_t = R\, g(t, d_{\text{model,in}})$ for the replicate construction.

In general $g(t, d_{\text{model,out}}) \neq R\, g(t, d_{\text{model,in}})$, so the witness cannot satisfy both the circuit equation and the desired replicated hidden state. The resulting mismatch at the first layer makes Attack B ZK-detectable (red cross in Figure 6).

RoPE (rotary encodings). RoPE rotates queries/keys according to frequency-specific angles. To preserve RoPE under Attack B we:

- use the same rotation frequencies in each replicated head; and
- apply RoPE identically in each replicated block of coordinates.

If the inner head uses rotation matrix R_t at position t, the outer model applies R_t independently in each block. Attention scores and probabilities are then identical across blocks and equal to those of A_{in}.

Takeaway. Learned tables and RoPE-style schemes can be aligned with both attacks by copying or replicating positional vectors in a way that respects Section 4.1. Deterministic sinusoidal encodings, when fixed as a public function of (t, d_{model}), block the widen-by-replication trick of Attack B, but leave Attack A and the learned/RoPE variants of Attack B intact. This is therefore a limitation of one particular construction rather than of Hollow-LLM more broadly. Fixed sinusoidal encodings prevent our width-expansion route, but they do not affect Attack A, and Attack B remains available under learned or RoPE-style positional schemes. Since many modern LLMs no longer use fixed sinusoidal encodings, this restriction narrows one variant without substantially reducing the practical relevance of the attack.

5. Evaluations

Beyond theoretical analysis, our evaluation aims to empirically corroborate the core claim of Hollow-LLM attack. Our experiments pursue three questions: (1) whether serve-side metrics (compute cost, latency, and memory cost) remain invariant when A_{in} is fixed, even as A_{out} changes; (2) whether zero-knowledge proving costs scale with the size of the declared model rather than the executed one; and (3) whether each structural transformation of attacks modifies only the symbolic declared circuit while leaving the executed computation invariant, and whether these transformations compose cleanly without introducing cross-effects.

Our evaluation focuses on serve/prove asymmetry and exact circuit preservation; deployment-level detectability is discussed separately in Section 6..

Evaluation Methodology. To answer these questions, we evaluate each configuration end-to-end: serving a request, generating a proof, and verifying the proof. For every declared architecture, we measure serve-side behavior, proving cost, and verification correctness while keeping the executed inner model fixed. By isolating the contributions of A_{in}, A_{out}, and their structural differences, the methodology directly tests the causal dependencies of our design.

5.1. Experimental Setup

Evaluation Model. Our design involves two models: (1) the *executed* inner model (A_{in}) during serving, which determines the computation used to serve a request, and (2) the *declared* outer model (A_{out}), which determines the public circuit used for zero-knowledge proving. Our evaluation makes this separation explicit by fixing A_{in} while systematically varying A_{out}. This serve/prove separation enabled by the Hollow-LLM attack is agnostic to the underlying model family and is expected to hold across a wide range of scale ratios (guidelines for choosing a ratio are discussed in Section 6). For evaluation, we adopt the ZKP procedure from zkGPT [7] as a concrete ZKP-verified LLM inference setting. Following the zkGPT framework, the prover executes a compact 6-layer, 512-dimensional Transformer, which we denote IM_1 as our baseline inner model. All serving-side costs, including FLOPs, memory traffic, and latency, are fully determined by this executed model. In contrast, the declared outer architecture is inflated through structural transformations that alter its symbolic shape while preserving its functional behavior. For this inner model, we instantiate a single representative scale ratio for Attack A, Attack B, and their composition A+B, introduced in Section 4:

- **Attack A (add-zero-layers):** increases the declared depth by inserting identity residual blocks (6 layers $\rightarrow$ 12 layers). The executed inner model, denoted IM_1, remains unchanged.
- **Attack B (enlarge-embedding):** widens the declared hidden and embedding dimensions (512 $\rightarrow$ 1024) while still executing the same IM_1 model.
- **A+B composition:** applies both depth and width expansion, yielding a declared architecture (12 layers, 1024 hidden) structurally equivalent to a larger transformer. Execution remains fixed at IM_1; only the declared circuit grows.

This fixed-A_{in}, varied-A_{out} design allows us to test the claim: serve-side metrics should track only the executed computation, while proving costs should scale only with the declared circuit.

Workloads and Testbed. We use fixed-length prompts of $T = 64$ tokens and greedy decoding. This choice intentionally isolates the structural effects of declared-model transformations: both Attack A and Attack B modify only the shape of the public circuit and are insensitive to prompt semantics, token distribution, or linguistic variation. Thus, short deterministic prompts are fully sufficient to exercise the serve/prove split that attacks aim to validate. We further use a compact transformer as the executed model A_{in}. This choice is deliberate: the purpose of our evaluation is to verify the *correctness and structural separation* of serving and proving, not to stress-test large-model throughput. Since transformations operate only on A_{out} and not on the runtime semantics of A_{in}, evaluating correctness does not require a large or instruction-heavy model. Using a small model enables complete end-to-end proving while keeping the

TABLE 2. **SERVE-SIDE AND ZKP COST UNDER DEPTH/WIDTH EXPANSIONS.** SERVE-SIDE TIME IS NORMALIZED TO THE BASELINE INNER MODEL IM_1 (6 LAYERS, 512-DIM). DECLARED PARAMETERS REFLECT THE PUBLIC CIRCUIT AFTER APPLYING ATTACK A/B/A+B. ZKP COSTS SCALE ONLY WITH THE DECLARED ARCHITECTURE, AS THE EXECUTED MODEL IS FIXED.

		Declared Params		Serve Time		ZKP Costs				Theoretical FLOPs	
Inner	**Attack**	**Layers**	**Embed**	**Prefill**	**Decode**	**Gates**	**Prover**	**Verify**	**Proof KB**	**Prefill**	**Decode**
IM_1	–	6	512	1×	1×	1×	1×	1×	45.6	1×	1×
IM_1	A	12	512	1×	1×	1.99×	1.69×	1.72×	83.5	1.4×	2.0×
IM_1	B	6	1024	1×	1×	2.95×	1.87×	1.30×	48.2	2.8×	3.9×
IM_1	A+B	12	1024	1×	1×	5.89×	4.81×	2.48×	88.3	4.5×	7.8×
IM_2	–	12	1024	2.4×	3.1×	5.89×	4.74×	2.56×	88.3	4.5×	7.8×

methodology controlled and fully reproducible. All experiments are conducted on a local workstation running Ubuntu 24.04 with Intel Core Ultra 7 265F CPU (20 cores).

5.2. Results

Our experiments vary the declared depth and width while keeping the executed inner model fixed, allowing us to isolate three properties of interest: (1) whether serve-side latency depends only on the executed model, (2) whether zero-knowledge proving cost scales with the declared architecture, and (3) whether structural manipulations exhibit locality and predictable composition.

Each row in Table 2 represents a pair of executed and declared architectures. The inner column denotes the model actually executed during serving (e.g., IM_1 is a 6-layer, 512-dimension GPT-2 backbone). The outer column specifies the declared public architecture obtained by applying Attack A (depth expansion), Attack B (width expansion), or their composition. These structural changes modify only the shape of the declared circuit; the executed inner model remains fixed for all rows. Serve-side columns report latency relative to the baseline. ZKP Costs columns report the ratio of gate size, prover time, and verification time induced by the declared model. Thus, serve-side ratios reflect properties of the executed model, whereas ZKP ratios reflect properties of the declared circuit. This separation makes it easy to read how each attack influences serving and proving, as discussed below.

Serve-side Invariance. Serve-side behavior is identical across all configurations because the attack always executes the same inner model. This is visible in Table 2: the Prefill and Decode columns remain at 1x for all A, B, and A+B rows, even though their declared models differ in depth and width. Only the honest large model shows higher serve-side ratios, confirming that serve-side cost depends solely on the executed model and is unaffected by changes to the declared architecture.

Declared-model Scaling. Zero-knowledge cost scales with the size of the declared architecture: increasing the declared depth or width enlarges the public circuit and raises gate count, proving time, and verification cost. In Table 2, Attack A and Attack B each increase different parts of the ZKP cost, where A adds depth-related overhead and B contributes width-related overhead. Their effects differ but are both strictly above the baseline. The A+B configuration combines both forms of expansion and therefore shows the largest ZK cost, even though the executed model remains fixed. Since the inner model never changes, all variation originates solely from the declared architecture.

Locality and Composition. Depth and width expansions influence separate parts of the declared circuit, and their zero-knowledge overheads accumulate predictably. In Table 2, Attack A increases only the Gates and proving cost associated with additional layers, while Attack B increases only the costs associated with wider embeddings. The A+B row combines both increases, matching the sum of the A and B effects and showing no change in serve-side columns, confirming that these expansions act locally and compose without interference.

Comparison to Honest Large Model. The full table shows that the A+B configuration reproduces the proving cost of the honest large reference model (IM_2, a 12-layer, 1024-dimensional Transformer). The A+B and IM_2 rows exhibit nearly identical gate counts, prover time, and verification time, indicating that the declared circuit obtained from A+B matches the structural complexity of a large model, even though the system continues to execute only the compact inner model IM_1.

This creates a compute asymmetry visible directly in the serve-side columns: IM_2 incurs substantially higher prefill and decode cost, while A+B remains at $1\times$ because it still executes IM_1. Thus, the attack exposes large-model proving complexity while avoiding the execution overhead of a large model.

Semantic Squivalence. All variants preserve the behavior of the executed model. Declared-model expansions introduce only structural changes to the public circuit and do not modify the inner model's forward computation, yielding identical outputs across all configurations. We confirmed this by observing unchanged perplexity across all experiments.

Summary of Findings. Together, these results demonstrate the attack's serve/prove separation: serve-side cost is fixed by the inner model, proving cost scales with the declared model, and structural expansions compose cleanly. Hollow-LLM attack can thus expose large-model proving complexity while executing only a compact model.

6. Discussion

6.1. Scope of the Attack

ZKP is Correct but an Overlooked Effort Gap. It is important to clarify that the Hollow-LLM attack does not imply any flaw in zero-knowledge proof technology, which remains fundamentally sound as a cryptographic primitive. In ZK LLM serve contexts, the proof mechanism correctly verifies that a given output is consistent with the computations defined by the declared model architecture. However, ZKPs do not attest to the actual effort or computational cost expended to produce that output during the procedure. The Hollow-LLM attack exploits this distinction: an adversary can supply a verifiably correct result (satisfying all equation-level checks) without performing the expected full-scale computation. This highlights the distinction between ensuring output correctness and ensuring execution effort. Any apparent overpromising stems not from the ZKP technology itself, but from deployment-layer assumptions. In particular, the mismatch arises when that commitment-relative guarantee is interpreted as evidence of commensurate execution effort, rather than from any failure of the proof system.

Choosing the Appropriate Outer Model Size. In principle, our ghost weight method allows the declared outer model size to be inflated arbitrarily without changing its input–output behavior. However, an overly aggressive enlargement can undermine stealth. A declared architecture that vastly exceeds what users expect for a given service or device may break immersion and raise suspicion. The attacker must therefore balance the cost-saving incentive of a larger outer model against the risk of detection through anomalous quality, performance, or deployment mismatch. At the same time, users cannot directly observe the true model size or workload, even benchmarking is not a reliable estimator of parameter count [12], [15], [28], [29], choosing the outer model size thus becomes a domain-specific perception-management problem. The declared outer model should remain plausible given the deployment context. For example, a billions-parameter flagship model might be credible in a cloud API from a major provider, whereas claiming the same size on a lightweight edge device would defy expectations. We do not claim that arbitrary over-claiming is indistinguishable. Extreme outer/inner gaps may create detectable quality shifts or behavioral anomalies. The practically relevant regime is modest over-claiming, where even relatively small savings can be economically meaningful at serving scale while producing weaker external evidence for detection.

Determining the exact threshold where enlargement breaches user trust is beyond the scope of the paper, as we focus on demonstrating that enlarged models with equivalent observable behavior can be constructed. Future work or operational policy must address how large an expansion can go undetected in various real-world scenarios.

Distinguishing Hollow-LLM Attack from Optimization Techniques. Optimization techniques such as KV caching reduce inference cost by reusing computations. For example, using a KV cache skips redundant recomputation of earlier tokens' effects. However, in benign optimization scenarios, the model preserves its advertised structure and behavior and is simply executed more efficiently. The Hollow-LLM attack, on the other hand, fundamentally violates this fidelity. Instead of accelerating a legitimate large model, the provider actually runs a different, smaller model while presenting it as a larger one. This means the system is no longer faithfully executing the advertised workload; it only mimics the outputs of the large model. Crucially, ZK proof-based verification will still pass because it checks only that the outputs are consistent with some execution of the declared architecture and does not verify that the full computational effort implied by that architecture was actually expended.

6.2. Countermeasures

One potential direction for mitigating the Hollow-LLM attack is to use zero-knowledge proofs not only to verify inference correctness but also to certify structural properties of the model, such as the sparsity or activity of its weight matrices. For example, a prover might demonstrate that certain matrices are not entirely zero. However, this approach faces two major limitations. First, adding sparsity-related subproofs for each parameter matrix introduces substantial overhead and undermines system scalability. Second, merely proving that weights are non-zero is insufficient to rule out Hollow-LLM constructions. An attacker may still be able to design ghost weights that satisfy non-triviality constraints while preserving the same low-effort computation path. A proof that no alternative ghost-weight constructions exist would be required, yet such a guarantee may be generally difficult to achieve.

A practical near-term strategy is to raise the adversary's cost, but it still cannot offer a fundamental security guarantee. For example, auditors can use behavioral techniques to raise the cost of cheating and increase detectability. These include challenge-based audits that introduce complex prompts only during verification, tests that temporarily disable attention heads or feedforward channels to measure ablation resilience, and diversity probes that evaluate output variability across decoding seeds or paraphrased prompts. Collectively, these techniques could stress-test the model's response surface and make low-effort deployments more risky. Another line of defense is to rely on hardware-based mechanisms such as Trusted Execution Environments (TEEs) instead of purely verification. For example, model training or inference can be executed inside a trusted hardware enclave, with hardware isolation and remote attestation providing stronger runtime monitoring and guarantees. Notably, with the recent trend that server-grade GPUs also support emerging GPU-TEE features [19], a combined CPU–GPU TEE pipeline can enforce that both the embedding flow and tensor-core computations are executed within measured, attested hardware. Nonetheless, TEEs still inherit limitations: they lack public verifiability, depend on proprietary hardware, and remain vulnerable to

side channels or firmware-level exploits. These constraints ultimately restrict their applicability in open, decentralized, or multi-party verification settings.

7. Related Work

Verifiable LLM Inference. Recent zero-knowledge machine learning (zkML) systems such as zkLLM [6], zkGPT [7], and ZkTorch [5] demonstrate the feasibility of verifying LLM inference via succinct proofs, even for models with tens of billions of parameters. For example, zkLLM proves a 13B-parameter model in under 15 minutes, and zkGPT verifies GPT-2 inference in roughly 25 seconds. General-purpose compilers such as ZKML and ZKTorch further extend verifiable inference to larger models through optimized circuits and proof aggregation. This trustless verification is especially advantageous in distributed or decentralized AI services, where third parties can independently validate the correctness of the model's outputs. Complementary model-oversight work such as WAVE [35] instead uses hardware-level observations to check whether observed execution is consistent with the claimed model, targeting deployment-level detection rather than the proof guarantee itself.

Alternatively, trusted execution environments (TEEs) are a popular and industry-ready solution. TEEs isolate model inference within secure hardware enclaves, enabling near-native latency because computation runs on dedicated hardware with minimal overhead. Earlier TEE-based verifiable AI frameworks are usually based on a CPU TEE [36], [37], such as Intel SGX. For example, Slalom [36] partitions neural network layers between an SGX enclave and a GPU, achieving 6× to 20× speedups for verifiable inference compared to enclave-only execution. More recently, NVIDIA released confidential GPU support [19], which provides a GPU TEE capability on the H100, B200, and RTX Pro 6000. GPU TEEs can confidentially execute LLM inference with modest performance overhead by encrypting data in use. These TEE-based methods require no code changes and provide real-time protected execution. However, unlike ZK proofs that offer cryptographic guarantees of correct execution, TEEs rely on specialized hardware and trust assumptions and do not provide public verifiability. Moreover, recent research show that TEE are inherently hardware-dependent and may be vulnerable to microarchitectural attacks [38], [39], [40], [41], [42], [43]. In scenarios that require strong trustless guarantees or open verification, ZK-proof–based approaches provide a more robust, although higher-latency, alternative.

Model-substitution detection and statistical auditing. Another related direction asks whether a remote service is actually serving the claimed model by comparing its behavior against external references or statistical fingerprints. Prior work such as VeriLLM, DetectLLM, SVIP [11], [44], [45] studies model integrity through output distributions, hidden-state or logit sampling, rank-based tests, equality-style checks, or side observations such as timing and traffic patterns. These methods are especially natural when the target model, its weights, or an official reference implementation are publicly available, because the auditor can compare the deployed service against known ground truth. Our setting is different. Hollow-LLM studies proprietary-weight zkLLM deployments in which the verifier sees only the public architecture, a commitment to private weights, and a proof transcript. In this commitment-only regime, behavioral auditing is best viewed as a complementary defense rather than part of the proof guarantee itself. Accordingly, our contribution is not to replace statistical verification, but to show that standard ZK verification alone does not rule out low-effort witnesses even when every proof remains valid.

Weight manipulation and inactive-layer constructions. Prior work has shown that carefully structured parameters can make part of a model's nominal computation ineffective. A particularly relevant example is [46], which studies federated learning and shows how model inconsistency can suppress client-side updates, weakening the intended privacy protection of secure aggregation. Our work follows a related intuition in the setting of zero-knowledge LLM inference: simple zeroed and structured ghost weights can preserve proof validity for a larger declared model while allowing the provider to carry out only the serve-time computation of a much smaller inner model. In this sense, our contribution is to make the same broad phenomenon concrete for zkLLM serving, where the central gap is not update privacy but the lack of effort binding between a valid proof and the computation actually performed.

8. Conclusion

In this paper, we identified and formalized an effort gap in ZK-verified LLM inference, showing that existing systems certify equation-level correctness for a declared architecture but remain agnostic to the computational effort actually expended. We instantiated this gap through the Hollow-LLM attack, in which a dishonest provider masks a smaller inner model behind ghost weights that preserve the public parameter count and zero-knowledge proofs while routing almost all computation through the cheaper model. We further developed concrete, training-free algebraic constructions that exploit transformer invariances to decouple advertised depth and width from true per-token cost, and we analyzed their security and economic impact in realistic deployment settings. Our findings underscore that proof validity alone is insufficient to guarantee faithful execution effort and challenge emerging narratives around zkML-based assurance.

Acknowledgment

The authors would like to thank Professor Jiapeng Zhang and Xinyu Mao for their insightful discussions and valuable comments. We also thank Haoxuan Xu for his feedback and support throughout the development of this project. Finally, we thank the anonymous reviewers and our shepherd for their constructive guidance during the revision process.

References

[1] H. Ge, Y. Li, Q. Wang, Y. Zhang, and R. Tang, "When backdoors speak: Understanding llm backdoor attacks through model-generated explanations," in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2025, pp. 2278–2296.

[2] Y. Li, H. Huang, Y. Zhao, X. Ma, and J. Sun, "Backdoorllm: A comprehensive benchmark for backdoor attacks and defenses on large language models," *arXiv preprint arXiv:2408.12798*, 2024.

[3] S. Li, H. Liu, T. Dong, B. Z. H. Zhao, M. Xue, H. Zhu, and J. Lu, "Hidden backdoors in human-centric language models," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 3123–3140.

[4] G. Sun, Z. Wang, B. Tian, M. Liu, Z. Shen, S. He, Y. He, W. Ye, Y. Wang, and A. Li, "Coin: Counting the invisible reasoning tokens in commercial opaque llm apis," *arXiv preprint arXiv:2505.13778*, 2025.

[5] B.-J. Chen, L. Tang, and D. Kang, "Zktorch: Compiling ml inference to zero-knowledge proofs via parallel proof accumulation," *arXiv preprint arXiv:2507.07031*, 2025.

[6] H. Sun, J. Li, and H. Zhang, "zkllm: Zero knowledge proofs for large language models," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 4405–4419.

[7] W. Qu, Y. Sun, X. Liu, T. Lu, Y. Guo, K. Chen, and J. Zhang, "zkGPT: An Efficient Non-interactive Zero-knowledge Proof Framework for LLM Inference," in *34st USENIX Security Symposium (USENIX Security 25)*, 2025.

[8] D. Pasquini, E. M. Kornaropoulos, and G. Ateniese, "{LLMmap}: Fingerprinting for large language models," in *34th USENIX Security Symposium (USENIX Security 25)*, 2025, pp. 299–318.

[9] H. Wang and T. Hoang, "ezdps: An efficient and zero-knowledge machine learning inference pipeline," *arXiv preprint arXiv:2212.05428*, 2022.

[10] B.-J. Chen, S. Waiwitlikhit, I. Stoica, and D. Kang, "Zkml: An optimizing system for ml inference in zero-knowledge proofs," in *Proceedings of the Nineteenth European Conference on Computer Systems*, 2024, pp. 560–574.

[11] Y. Sun, Y. Li, Y. Zhang, Y. Jin, and H. Zhang, "Svip: Towards verifiable inference of open-source large language models," *arXiv preprint arXiv:2410.22307*, 2024.

[12] W. Cai, T. Shi, X. Zhao, and D. Song, "Are you getting what you pay for? auditing model substitution in llm apis," *arXiv preprint arXiv:2504.04715*, 2025.

[13] X. Zhu, Y. Ye, T. Qiu, H. Zhu, S. Tan, A. Mannan, J. Michala, R. A. Popa, and W. Neiswanger, "Auditing black-box llm apis with a rank-based uniformity test," *arXiv preprint arXiv:2506.06975*, 2025.

[14] M. J. Yuan, C. Lospoy, S. Lai, J. Snewin, and J. Long, "Trust, but verify," 2025. [Online]. Available: https://arxiv.org/abs/2504.13443

[15] I. Gao, P. Liang, and C. Guestrin, "Model equality testing: Which model is this api serving?" *ICLR*, 2025.

[16] S. Alhazbi, A. Hussain, G. Oligeri, and P. Papadimitratos, "Llms have rhythm: Fingerprinting large language models using inter-token times and network traffic analysis," *IEEE Open Journal of the Communications Society*, 2025.

[17] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," *arXiv preprint arXiv:1806.03287*, 2018.

[18] T. Hunt, C. Song, R. Shokri, V. Shmatikov, and E. Witchel, "Chiron: Privacy-preserving machine learning as a service," *arXiv preprint arXiv:1803.05961*, 2018.

[19] NVIDIA, "Confidential Compute on NVIDIA Hopper H100," https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/HCC-Whitepaper-v1.0.pdf, 2023.

[20] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International conference on machine learning*. PMLR, 2016, pp. 201–210.

[21] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim *et al.*, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *iEEE Access*, vol. 10, pp. 30 039–30 054, 2022.

[22] J. Lee, E. Lee, J.-W. Lee, Y. Kim, Y.-S. Kim, and J.-S. No, "Precise approximation of convolutional neural networks for homomorphically encrypted data," *IEEE Access*, vol. 11, pp. 62 062–62 076, 2023.

[23] W. Jin, Y. Yao, S. Han, J. Gu, C. Joe-Wong, S. Ravi, S. Avestimehr, and C. He, "Fedml-he: An efficient homomorphic-encryption-based privacy-preserving federated learning system," *arXiv preprint arXiv:2303.10837*, 2023.

[24] F. Effendi and A. Chattopadhyay, "Privacy-preserving graph-based machine learning with fully homomorphic encryption for collaborative anti-money laundering," in *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 2024, pp. 80–105.

[25] M. Steyvers, H. Tejeda, A. Kumar, C. Belem, S. Karny, X. Hu, L. W. Mayer, and P. Smyth, "What large language models know and what people think they know," *Nature Machine Intelligence*, vol. 7, no. 2, pp. 221–231, 2025.

[26] C. A. Lehmann, C. B. Haubitz, A. Fügener, and U. W. Thonemann, "The risk of algorithm transparency: How algorithm complexity drives the effects on the use of advice," *Production and Operations Management*, vol. 31, no. 9, pp. 3419–3434, 2022.

[27] A. McConnon. (2024, July) Are bigger language models always better? IBM Think Blog. [Online]. Available: https://www.ibm.com/think/insights/are-bigger-language-models-better

[28] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[29] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[30] Ledger. (2025, 3) Zero-knowledge machine learning. Ledger Academy. "This certificate contains details such as model size and parameters, confirming that a certain computation has been done while hiding sensitive information from the verifiers". Accessed 8 Nov 2025. [Online]. Available: https://www.ledger.com/academy/glossary/zero-knowledge-machine-learning-zkml

[31] CoinGecko. (2024, 1) zkml (zero-knowledge machine learning). CoinGecko. "zkML provides a cryptographic certificate verifying an ML model's inference, including details like model size and parameters, ensuring that a prompt has been executed without revealing sensitive data". Accessed 8 Nov 2025. [Online]. Available: https://www.coingecko.com/en/glossary/zkml

[32] C. Murray and R. Williams, "Circuit lower bounds for nondeterministic quasi-polytime: an easy witness lemma for np and nqp," in *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, 2018, pp. 890–901.

[33] R. Impagliazzo, V. Kabanets, and A. Wigderson, "In search of an easy witness: Exponential time vs. probabilistic polynomial time," *Journal of Computer and System Sciences*, vol. 65, no. 4, pp. 672–694, 2002.

[34] A. Kattis and J. Bonneau, "Proof of necessary work: Succinct state verification with fairness guarantees," in *International Conference on Financial Cryptography and Data Security*. Springer, 2023, pp. 18–35.

[35] H. Xu, C. Gong, B. Liu, H. Zheng, B. Chen, and M. Li, "Wave: Leveraging architecture observation for privacy-preserving model oversight," ser. ASPLOS '26. New York, NY, USA: Association for Computing Machinery, 2026, p. 2212–2231. [Online]. Available: https://doi.org/10.1145/3779212.3790247

[36] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," *arXiv preprint arXiv:1806.03287*, 2018.

[37] H. Hashemi, Y. Wang, and M. Annavaram, "Darknight: An accelerated framework for privacy and integrity preserving deep learning using trusted hardware," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 212–224.

[38] S. Van Schaik, A. Seto, T. Yurek, A. Batori, B. AlBassam, D. Genkin, A. Miller, E. Ronen, Y. Yarom, and C. Garman, "Sok: Sgx. fail: How stuff gets exposed," in *2024 IEEE symposium on security and privacy (SP)*. IEEE, 2024, pp. 4143–4162.

[39] D. Moghimi, "Downfall: Exploiting speculative data gathering," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 7179–7193.

[40] S. Van Schaik, A. Kwong, D. Genkin, and Y. Yarom, "Sgaxe: How sgx fails in practice," 2020.

[41] J. Van Bulck and F. Piessens, "Sgx-step: An open-source framework for precise dissection and practical exploitation of intel sgx enclaves," in *39th Annual Computer Security Applications Conference (ACSAC)*. ACM, 2023.

[42] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 142–157.

[43] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter, "Leaky cauldron on the dark land: Understanding memory side-channel hazards in sgx," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 2421–2434.

[44] K. Wang, Z. Zhao, X. Song, Z. Li, L. Xia, C. Tong, B. Shi, W. Qu, E. Yang, and L. Ai, "Verillm: A lightweight framework for publicly verifiable decentralized inference," *arXiv preprint arXiv:2509.24257*, 2025.

[45] J. Su, T. Zhuo, D. Wang, and P. Nakov, "Detectllm: Leveraging log rank information for zero-shot detection of machine-generated text," in *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023, pp. 12 395–12 412.

Ethics considerations

Our work is a methodological and systems-oriented study of zero-knowledge verified LLM inference. We formalize the Hollow-LLM attack, construct algebraic ghost-weight families, and evaluate their impact using self-hosted transformer models and zkML pipelines in a controlled research setting; we do not involve human subjects, user studies, or any form of personal or sensitive data. The main ethical concern is that our constructions could be abused by a dishonest provider in practice to under-provision compute while still passing zero-knowledge verification. However, to the best of our knowledge, no company offers commercial real-time ZKP LLM inference verification service. Making the attacks explicit before real-world deployments emerge is intended to help standardize defenses and evaluation practices, not to enable threats.

Conflict of Interest Statement

The authors declare that they have no competing financial or non-financial interests related to this work.

LLM usage considerations

- **Originality.** LLMs were used for editorial purposes in this manuscript, and all outputs were inspected by the authors to ensure accuracy and originality. All core technical contributions (formal definitions, attack constructions, proofs, threat model, experiments) and the literature review were developed and written by the authors.
- **Transparency.** LLMs are not part of the paper's methodology: they were not used to design attacks, implement systems, or generate or analyze results. Reproducibility of our findings does not depend on access to any particular LLM service.
- **Responsibility.** LLM usage was limited to light text editing and did not involve training new models, collecting additional data, or processing sensitive or proprietary content.

Appendix A.
Meta-Review

The following meta-review was prepared by the program committee for the 2026 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

Summary

This paper identifies an "effort gap" in current Zero-Knowledge Machine Learning (zkML) protocols, where proofs verify mathematical correctness but not the actual computational work performed. The authors introduce the "Hollow-LLM Attack," using elegant algebraic constructions called "ghost weights" to allow a provider to serve a small model while providing valid zero-knowledge proofs for a much larger one.

Scientific Contributions

- Identifies an Impactful Vulnerability.
- Provides a Valuable Step Forward in an Established Field.

Reasons for Acceptance

1) **Identifies an Impactful Vulnerability:** The paper formalizes a novel security flaw where a dishonest provider can bypass the intended resource costs of LLM serving. By demonstrating that a valid proof does not necessarily imply a "Proof of Computation," the authors reveal a significant economic and security risk for verifiable outsourced inference.
2) **Provides a Valuable Step Forward in an Established Field:** The work provides a necessary bridge between theoretical cryptography and practical machine learning security. The proposed "ghost weight" constructions (Attacks A and B) are technically sound and articulately explained, offering a principled way to evaluate the limitations of existing Transformer-based zkML schemes.

Noteworthy Concerns

1) The motivation for inflating the declared model size is weak. In many settings, users evaluate models based on output quality rather than parameter count. If the gap between claimed and actual model size is large, the attack may become detectable; if the gap is small, the economic benefit may be limited. The paper lacks a more detailed discussion of realistic attacker incentives and cost savings.