Asymmetric-Knowledge Zero-Knowledge for Autoregressive Language Model Inference (v2)

PSI-LM, AKZK, AsymVZK, and VEWC — extended empirical study with bootstrap CIs and tightened proofs

Einar Holt, Founder & Partner at tenki

May 2026 (v2)

Table of Contents

Version note. This is v2.1 of the May 2026 release. v1 reported a small-N proof-of-concept on Qwen 2.5 and Qwen 3.5 with proof sketches and CPU-bound

experiments. v2 (May 2026) kept the construction unchanged but: (a) reran the empirical sweep at $N = 500$ trials per cell on GPU with bootstrap 95% confidence intervals; (b) replaced three "proof sketches" with full hybrid arguments in §6.4, §7.5, and §8.2; (c) tightened the abstract claims; (d) added visual figures. **v2.1 (citation patch)** adds Chong, Ohsaki and Ng (2025) [@chong2025tractable] as a closely related concurrent thread on non-cryptographic asymmetric verification of LLM outputs (§3.4); no construction or empirical numbers change between v2 and v2.1. The two limitations explicitly *deferred to a follow-up paper* remain: integration of a real per-position SNARK in place of the consistency stub (§9.9) and a formal black-box separation between AKZK and standard ZK (§5.6). All other claims in v1 are reproduced or strengthened by the larger benchmark.

1 Abstract

We present three constructions for verifiable autoregressive language-model inference and a new formal threat model, **asymmetric-knowledge zero-knowledge** (AKZK), which is essential to the second and third of them.

Background. Existing systems for zero-knowledge proofs over large language models, typified by zkLLM (Sun et al., CCS 2024) and zkGPT (Qu et al., USENIX Security 2025), prove one forward pass at a time and report costs in the order of fifteen minutes per token on 13B models. Autoregressive generation, where a chat completion requires hundreds of sequential forward-pass proofs, has not been addressed end to end. Naive composition costs roughly ten days per response.

Contribution 1: PSI-LM (baseline). We formalize a sampling-soundness construction that decouples prover time from sequence length. The prover commits to per-token logits; the verifier challenges a small random subset of positions via Fiat-Shamir; soundness follows from a union bound over the challenge distribution. Positional zero-knowledge holds outside the challenge set.

Contribution 2: the AKZK threat model. We formalize an extension of the Goldwasser-Micali-Rackoff framework where the verifier has access to a public deterministic auxiliary oracle that it can evaluate locally. Standard zero-knowledge corresponds to the trivial-oracle case. We give a simulator-based definition for protocols in this model. A formal black-box separation between AKZK and standard ZK is left to a follow-up paper.

Contribution 3: AsymVZK (main construction). We instantiate AKZK with the oracle $\mathcal{O}_{M_d}$, a small draft model the verifier runs freely. The prover runs speculative decoding (Leviathan et al., 2023) under a cryptographically committed accept/reject test. The verifier replays the draft model at every position (free relative to cryptographic proofs) and only challenges a small number of *accepted* and *rejected* positions with full zero-knowledge proofs. Verifier work scales with disagreement between M and M_d, not with sequence length.

Contribution 4: VEWC (verifiable entropy-weighted challenges). Cut-and-choose has a structural weakness: a content-targeted adversary that lies at a single semantically loaded token is detected only with probability k/L. We introduce a *semantic-budget adversary* class, a *verifiable entropy commitment* primitive, and prove a new soundness theorem: with entropy-weighted challenges, undetected probability is bounded by $(1 - B/H_{\text{tot}})^k$ where B is the budget and H_{tot} is total Shannon entropy.

Empirical (v2 GPU sweep, $N = 500$ per cell, 95% bootstrap CIs). We implement all three constructions and evaluate on **Qwen 2.5-1.5B + Qwen 2.5-0.5B** for AsymVZK and on **Qwen 3.5-4B + Qwen 3.5-0.8B** (released March 2026) for VEWC. Speculative-acceptance rate is about 70-86 percent across five prompt domains, meaning AsymVZK invokes cryptographic proofs at roughly a quarter of positions. Empirical detection rates against a content-targeted adversary track the theoretical $1 - (1 - B/H_{\text{tot}})^k$ bound for VEWC; the gap to AsymVZK uniform is over an order of magnitude at low k and remains substantial at k where AsymVZK saturates. The honest cost of the protocol is also visible in the data: against an adversary that targets *low*-entropy positions VEWC degrades to zero detection at $k \leq 32$, motivating the hybrid configuration discussed in §9.7.3.

Speedup framing. PSI-LM and AsymVZK both achieve $\sim 32\times$ prover speedup over the naive autoregressive zkLLM baseline at $L = 1024$, $k = 32$. AsymVZK does *not* speed up the prover further over PSI-LM; its added value is on the verifier side (free draft-model replay gives semantic-content soundness coverage) and in the AKZK threat model itself.

Positioning. AsymVZK and VEWC do not replace full-trace ZKML. They are complementary protocol frames that exploit a new threat model. The AKZK framework, where the verifier is armed with weak public model knowledge, has applications beyond LLMs: verifiable federated learning, streaming inference, and edge-deployed models with retracing.

Reproducibility. All code, the speculative-decoding harness, and the seven CSV result files in §9 are public. The tenki-research GitHub artifact and IACR ePrint version are referenced in §11. Every figure in §9 is regenerated from the CSV via `experiments/make_paper_graphs.py`.

---

2 1. Introduction

The trust assumption that a language-model response was produced by a specific model running correctly is increasingly a regulatory requirement. Frameworks such as the EU AI Act, DORA (Digital Operational Resilience Act), and medical-supervision guidelines all assume traceability of machine-learning decisions that affect individuals. The relevant cryptographic primitive for *transferable* third-party verification is the zero-knowledge proof: a proof anyone can check, with very low verifier cost, without trusting the service provider's infrastructure.

In Nordic context the need is most visible in three sectors. The healthcare sector, through the Norwegian Helseplattformen debate and the broader discussion of AI in clinical journaling, has become explicit about model decisions needing to be auditable. Banking and insurance under DORA require audit trails for algorithmic decisions. Public administration, under Datatilsynet's interpretation of GDPR Article 22, requires explanations for automated decisions. In each setting the operative requirement is *transferable* verifiability: a regulator must be able to confirm that a model was used correctly without re-executing it and without access to either model weights or user prompt.

2.1 1.1 The gap the literature has not closed

The dominant research thread in ZKML, typified by zkLLM (Sun, Li and Zhang, CCS 2024), zkGPT (Qu et al., USENIX Security 2025), zkPyTorch (Polyhedra, 2025), and DeepProve (Lagrange Labs, 2025), has focused on compiling LLM inference directly into an arithmetic circuit and proving the entire forward-pass computation. The results are impressive: zkGPT proves one GPT-2 forward pass in under 25 seconds; zkLLM reaches 13-billion-parameter models in under 15 minutes per forward pass with proofs under 200 KB and 1 to 3 second verification.

A fundamental gap remains open: **all of these systems measure the cost of one forward pass on one prompt**. Autoregressive generation, where the model produces one token at a time conditioned on the growing context, does not compose for free. To our knowledge, no production system has reported end-to-end proofs for an autoregressive chat completion of realistic length. A naive composition costs $L \cdot T_{\text{zk}}$, that is $L \cdot 15$ minutes for L-token output on 13B models, which lands around ten days for typical 1000-token responses.

2.2 1.2 Our approach, in three steps

Step 1: PSI-LM, sampling-soundness as baseline. We formalize a cut-and-choose construction where the prover commits to logit vectors per token and the verifier challenges a random subset of k positions via Fiat-Shamir. Soundness follows from a union bound over the challenge distribution: an adversary lying on fraction ε of positions is detected with probability $1 - (1 - \varepsilon)^k$. The construction decouples prover time from L: the prover's cost is $O(L \cdot T_{\text{infer}} + k \cdot T_{\text{zk-pos}})$. For $k = 32$ and $L = 1024$ this delivers about a $32\times$ speedup over the naive autoregressive composition.

Step 2: AsymVZK, asymmetric-knowledge as a new threat model. We observe that PSI-LM and the entire existing ZKML literature assume the verifier has zero model knowledge. Break that assumption and the cost structure collapses. AsymVZK formalizes a new threat model, *asymmetric-knowledge zero-knowledge* (AKZK), where the verifier has access to a public auxiliary oracle $\mathcal{O}$. We instantiate this with a *draft model* M_d that the verifier may run freely. The protocol uses speculative decoding (Leviathan et al., 2023) under a cryptographically committed accept/reject test. The verifier replays the draft model at

every position (free relative to cryptographic proofs) and only challenges a small number of accepted and rejected positions with full zero-knowledge proofs.

Step 3: VEWC, semantic-content soundness via entropy weighting. Even with PSI-LM and AsymVZK, cut-and-choose remains weak against a single-token semantic lie. VEWC introduces a new adversarial class (semantic-budget), a new soundness theorem (entropy-weighted), and a new primitive (verifiable entropy commitments). The result is that single-token semantic attacks become exponentially easier to catch on real models.

2.3 1.3 Contributions, named explicitly

1. **Asymmetric-knowledge zero-knowledge (AKZK)**, a new formal threat model. We give a simulator-based definition (§5) that generalizes the classical Goldwasser-Micali-Rackoff framework, and show that standard ZK is the trivial-oracle case.

2. **PSI-LM** (§6), a sampling-soundness construction as baseline for comparison. Soundness $1 - (1 - \varepsilon)^k$ with explicit ε-δ tradeoff; positional zero-knowledge.

3. **AsymVZK** (§7), the first zero-knowledge construction for autoregressive LLM inference that uses speculative decoding cryptographically. Hybrid soundness $1 - (1 - \varepsilon_a)^{k_a}(1 - \varepsilon_r)^{k_r}$ over accepted/rejected challenges; verifier cost scales with disagreement, not with sequence length.

4. **VEWC** (§8), the third construction, with a new theorem (entropy-weighted soundness), a new primitive (verifiable entropy commitments), and an empirical demonstration on Qwen 3.5 of strict improvement over uniform cut-and-choose against content-targeted adversaries.

5. **Implementation on real models** (§9). We run all three constructions on Qwen 2.5-1.5B-Instruct (large) and Qwen 2.5-0.5B-Instruct (draft), and on Qwen 3.5-4B + Qwen 3.5-0.8B for VEWC validation; we measure acceptance rates across prompt domains, sweep prover/verifier costs over the (L, k) grid, and validate empirical detection rates against theoretical bounds at $N = 500$ trials per cell with bootstrap 95% confidence intervals.

6. **Positioning and future work** (§11): the AKZK framework opens a family of constructions beyond AsymVZK and VEWC, including verifiable federated learning, streaming inference, and edge-deploy with retracing.

2.4 1.4 What is *not* in scope of this paper

We are explicit about three limitations because each materially affects how the rest of the paper should be read.

The per-position consistency proof Π_M is stubbed in our reference implementation. The verifier re-evaluates the model at challenged positions instead of running a true zero-knowledge proof system at that step. Every wall-clock projection in §9.8 substitutes

published zkLLM (Sun et al., 2024) cost figures for the inner Π_M proof. The protocol overhead numbers we measure are real; the end-to-end zero-knowledge service-cost numbers are extrapolations. Production integration with a zkLLM-class proving stack is left to a follow-up paper. We say so again at the start of §9.

A formal black-box separation between AKZK and standard ZK is not given here. §5.6 introduces "substantive AKZK" and argues by appeal that no classical ZK protocol for the same problem family achieves the same asymptotic verifier cost; the lower bound itself is left to a follow-up. The §5.5 reductions establish only backward compatibility (standard ZK ⇔ trivial-oracle AKZK).

The accept/reject pattern $(a_1, \ldots, a_L)$ is publicly committed by AsymVZK and VEWC. This is a real leak about prompt-model interaction; for medical, legal, and financial deployments where the prompt itself encodes sensitive information, the leak may be unacceptable. §10.2 lists three candidate mitigations (stronger draft model, differentially-private noising, threshold-committed accept bits); none is implemented. The threshold-committed variant is, in our view, the right next primitive to design.

A reader who cares mainly about the threat-model novelty can skip to §5; one focused on the empirical claims should treat §9.8 as a projection rather than a measurement and keep §1.4 in mind.

---

3 2. Background

This section assembles what the reader needs from four areas: zero-knowledge proofs, language-model architecture, the ZKML cost regime, and speculative decoding as an inference-acceleration technique.

3.1 2.1 Zero-knowledge proofs

A zero-knowledge proof system consists of a prover P and a verifier V running a protocol to decide whether a statement x belongs to a language $\mathcal{L}$. Three properties are required:

- **Completeness**: if $x \in \mathcal{L}$ and P knows a witness w such that $(x, w) \in R$ for relation R, then V accepts with high probability.
- **Soundness**: if $x \notin \mathcal{L}$, then V accepts with negligible probability, even against adversarial provers.
- **Zero-knowledge**: V learns nothing from the protocol beyond the fact that $x \in \mathcal{L}$. Formally, there exists a simulator Sim that produces a transcript that is computationally indistinguishable from a real protocol transcript, given only x.

The classical formulation is interactive (Goldwasser, Micali and Rackoff, 1989). The Fiat-Shamir transform (Fiat and Shamir, 1986) makes it non-interactive by replacing the verifier's random challenges with a hash of prior messages, secure in the random-oracle model.

For LLM applications we use building blocks like commitment schemes: a prover binds itself to a value m without revealing it and may later open the commitment. *Hiding* ensures m is not revealed before opening; *binding* ensures the commitment cannot be opened to two different values. Pedersen commitments give both hiding and binding under the discrete-log assumption; SHA-256-based commitments give only computational binding but are simpler and used in our reference implementation.

For matrix-heavy ML computations, the **sumcheck protocol** (Lund, Fortnow, Karloff and Nisan, STOC 1990) has become the preferred primitive: it lets the prover convince the verifier of a sum over a multilinear extension of a tensor in polylog rounds. Sumcheck generalizes the Freivalds algorithm (Freivalds, 1979) to arbitrary polynomials. **Lookup arguments**, including plookup, Lasso (Setty et al., 2023), and Jolt (Arun et al., 2023), make non-linearities (softmax, GELU, LayerNorm) tractable by letting the proof reference a precomputed table, with prover work scaling in the number of *unique* lookups actually used. **Folding schemes** (Nova, SuperNova, HyperNova; Kothapalli et al., 2021-2023) let the prover accumulate identical repeated computations incrementally into a single accumulator.

3.2 2.2 Language-model architecture in brief

A modern LLM is a decoder-only transformer (Vaswani et al., 2017). A prompt $p = (x_1, \dots, x_n)$ is tokenized into IDs from a vocabulary $\mathcal{V}$ (typically 30,000 to 150,000 tokens). Each token maps to a vector via an embedding table, and the sequence passes through L_d identical blocks (typically 12 to 80 depending on model size). Each block consists of a self-attention module and a feed-forward network (FFN), wrapped in residual connections and layer normalization.

For generation the model runs *autoregressively*: given prompt p, it computes logits $z_{n+1} \in \mathbb{R}^{|\mathcal{V}|}$ for the next token, samples a token y_1 according to a sampling rule R (greedy, top-k, top-p, or temperature-scaled categorical), then computes z_{n+2} based on (p, y_1), and so on. The sequence $h_t = (p, y_1, \dots, y_{t-1})$ is the context at position t.

Operators that are problematic cryptographically include **softmax** (exponential, division, global normalization), **GELU**, **SiLU**, and **SwiGLU** (transcendental non-linearities), **LayerNorm/RMSNorm** (square roots, divisions), and **self-attention** (quadratic in sequence length). Matrix multiplication is the expensive operation in floating-point arithmetic but is relatively cheap in a sumcheck-based SNARK framework.

3.3 2.3 Where the intersection is hard

LLMs operate over floating-point numbers (typically fp16 or bf16); arithmetic circuits operate over a finite field $\mathbb{F}_p$. Direct emulation of one fp32 multiplication costs hundreds to thousands of constraints. Production ZKML systems therefore quantize to INT8 or INT16 and pay a measurable perplexity cost. The truncation and rounding checks that come with quantization are, according to zkGPT (Qu et al., 2025), the single largest contributor to prover time, not the matrix multiplications themselves.

Concrete scale: one GELU evaluation is reported at about 11,654 PlonK constraints with polynomial approximation. The zkAttn construction in zkLLM decomposes softmax into K base-b segments and applies tlookup to each. The self-attention matrix, $N \times N$ for sequence length N, must in a classical plonkish setup be committed in full; for GPT-2 small with $N = 2048$, $L_d = 12$ layers, and 12 heads, that gives roughly 600 million cell witnesses just for the attention scores.

There is one bottleneck the literature has not pushed hard on: witness-generation memory under autoregressive decoding. Each new token requires a fresh forward pass on growing context. The KV-cache (the key/value buffer that plaintext LLM serving uses to avoid redundant attention) has no clean analog in ZKML: cached tensors must still be committed and proven. This is the gap PSI-LM and AsymVZK both address.

3.4 2.4 Speculative decoding as plaintext acceleration

Speculative decoding (Leviathan, Kalman and Matias, ICML 2023; Chen et al., 2023) is a plaintext LLM acceleration technique that we adopt cryptographically in AsymVZK. The idea: let a smaller, cheaper "draft" model M_d propose tokens, and let the larger M verify the proposals through an accept/reject test that *preserves the marginal distribution* of M.

Concretely, at each position t the draft samples a proposal y_t^{draft} from its distribution. The acceptance test is:

$$u \sim U(0,1), \quad \text{accept if } u \leq \frac{p_M(y_t^{draft} \mid h_{t-1})}{p_{M_d}(y_t^{draft} \mid h_{t-1})}.$$

If accepted: $y_t = y_t^{draft}$. If rejected: y_t is sampled from the residual distribution $\max(p_M - p_{M_d}, 0)$, normalized. The correction ensures the marginal of y_t is exactly p_M, so speculative decoding is unbiased.

In practice it predicts several tokens at a time ("draft chunk") and the large model evaluates them in parallel before applying accept/reject. The plaintext speedup is 2-3x on well-aligned (M, M_d) pairs, because M runs batched inference rather than autoregressively.

For our purposes the prover-side speedup is irrelevant; what matters is **that the accept/reject decision is a pure function of** $(p_M(y^{draft}), p_{M_d}(y^{draft}), u)$ **and is therefore cryptographically checkable**: if the verifier holds M_d and u, and the prover claims a value for $p_M(y^{draft})$, the verifier can determine whether the accept/reject decision is consistent with that claim. This structural property is what makes AsymVZK possible.

4 3. Related Work

We synthesize here what we view as the five most relevant threads in the ZKML literature, viewed through the lens of our constructions.

4.1 3.1 ZKML systems for transformers

zkLLM (Sun et al., 2024) is the most important precursor. It combines tlookup (a parallelized lookup argument over tensor-valued lookups) with zkAttn (a custom sumcheck for self-attention) and runs as CUDA kernels directly on GPU memory. The result is one forward pass of LLaMA-2-13B in under 15 minutes, with proofs under 200 KB and 1-3 second verification. The zkLLM paper explicitly states the construction measures only one forward pass; end-to-end autoregressive decoding is not benchmarked.

zkGPT (Qu et al., 2025) dramatically improves the cost profile for GPT-2 scale models: a forward pass of GPT-2 (117M parameters) is proven in under 25 seconds via *constraint fusion* for non-linear layers and *circuit squeeze* for parallelization. zkGPT confirms that the bottleneck has moved from matrix multiplication to rounding checks under quantization.

zkPyTorch (Polyhedra, 2025) is a compiler from PyTorch to the Expander proof system, reporting about 150 seconds of prover time per token for Llama-3.1-8B. Hardware and proof sizes are not publicly disclosed. **DeepProve-1** (Lagrange Labs) claims under one minute per inference on a 10-machine distributed setup. **ZKTorch** reports GPT-J 6B at about 20 minutes with 64 threads.

Modulus Labs published an early reference point: GPT2-XL (1.5 billion parameters) proven end-to-end on Halo2 in March 2024, with about 200 hours of prover time on a 128-core machine with 1 TB RAM and 10 TB disk. The reported overhead vs. plaintext inference is 1,125,761x.

A common feature of all these systems: the proof size and prover time apply to *one* forward pass. None reports end-to-end cost for a realistic autoregressive chat completion.

4.2 3.2 Proof systems

The consensus on where SNARKs for transformers should be built has converged: **sumcheck** (Lund et al., Libra) over multilinear extensions of tensors inside each layer, **lookup arguments** (Lasso, Jolt, LogUp) for non-linearities, optionally a **folding scheme** (Nova, HyperNova, Sangria) for repeated structure, and a KZG or Groth16 wrap on the outside for compact on-chain proofs. The 35.7x speedup zkLLM achieves over Halo2-based ZKML is empirical evidence that *Plonkish*-monolithic circuits are the wrong architecture.

For our constructions the most interesting observation is that folding schemes are structurally ideal for transformers (each of L_d identical blocks is one fold step) but folding of lookup arguments is still an open research problem (Bünz and Chen, 2024; Origami,

Mova, Protostar). This means the asymptotically “right” architecture for transformers is not yet a deliverable construction.

4.3 3.3 Adjacent privacy approaches

Three alternatives to ZK compete for the same services but solve different problems:

FHE (fully homomorphic encryption): runs the model on encrypted input. Zama Concrete-ML has shown that moving a *single attention layer* of phi-1.5 to FHE incurs about 2.5 seconds of latency per token; GPT-2 small end-to-end in FHE runs at about 11 seconds per token on GPU. No FHE stack has appeared on models larger than approximately 1B as of 2026.

MPC (secure multi-party computation): splits model and/or input across non-colluding parties. PUMA gives LLaMA-7B inference in about 5 minutes per token via 3PC; BumbleBee and SIGMA are comparable. Requires a non-collusion assumption that is awkward in deployment.

TEEs (trusted execution environments): NVIDIA H100 Confidential Computing reports under 7 percent overhead at frontier scale. Apple Private Cloud Compute and Anthropic Confidential Inference build on this. TEEs are clearly the winner on *runtime confidentiality*. However, TEE.fail (October 2025) demonstrated sub-$1000 DDR5 attacks that break Intel TDX/SGX and AMD SEV-SNP, and Heracles (CCS 2025) demonstrated a chosen-plaintext side channel against SEV-SNP. Both hardware vendors declared physical attacks out of scope.

Implication: ZK does not compete with TEE on runtime confidentiality. ZK delivers transferable cryptographic verifiability, a property none of the alternatives provide.

4.4 3.4 Slalom and the Freivalds tradition

Slalom (Tramèr and Boneh, ICLR 2019) used Freivalds’ algorithm (Freivalds, 1979) inside an SGX enclave to verify matrix multiplications outsourced to an untrusted GPU. Although Slalom is not cryptographic zero-knowledge, it is the most important reference point for *probabilistic ML verification*. The sumcheck protocol (Lund et al., 1990; Goldwasser, Kalai and Rothblum, STOC 2008) can be viewed as a generalization of Freivalds.

PSI-LM takes a similar step: we use probabilistic soundness over *positions* in an autoregressive sequence, where Slalom used it over rows/columns of a matrix multiplication. AsymVZK goes further: it uses Slalom-like asymmetry (verifier holds a *locally executable* lower-tier oracle) but replaces SGX with a cryptographically committed local model. The structure is similar; the substance differs.

A concurrent and closely related line is **Chong, Ohsaki and Ng (2025)** [@chong2025tractable], which proposes tractable asymmetric verification for LLM outputs by exploiting *deterministic replicability*: in a homogeneous hardware/software environment, autoregressive models are bit-exactly reproducible, so a validator can re-run

small random output segments and compare. They report verification over 12x faster than full regeneration with tunable detection probability, on simulated workloads. This sits in the same "asymmetric effort" genus as AsymVZK but in a non-cryptographic setting: there is no binding commitment, the verifier executes the *full* model on the audited segment (just on a strict subset of positions), and soundness rests on the homogeneity of the validator's stack rather than on a cryptographic primitive. AsymVZK and PSI-LM therefore complement rather than compete with this thread: Chong et al. give a deployment-friendly trust mechanism inside a single homogeneous fleet; we give a cryptographically binding mechanism that survives a heterogeneous, adversarial verifier-prover pair and produces a transferable proof.

4.5 3.5 Verifiable evaluation

South et al. (2024) propose verifiable evaluations: cryptographic proofs that a given model achieves a given accuracy on a held-out test set. This is an output-based approach in the same spirit as PSI-LM, but applied to batch evaluation rather than autoregressive inference. Our contribution can be viewed as continuing this line into generative settings.

4.6 3.6 Speculative decoding

Speculative decoding (Leviathan et al., 2023; Chen, Borgeaud et al., 2023) is a plaintext acceleration technique for LLM serving. Standard acceleration uses the draft model to predict several tokens in parallel, which the large model verifies in batched fashion. A 2-3x speedup is typical for well-aligned (M, M_d) pairs. To our knowledge no prior work has used the speculative-decoding accept/reject test as a *cryptographic* commitment. AsymVZK does.

4.7 3.7 The autoregressive gap, made explicit

To make clear why we view the autoregressive gap as central, we contrast the published numbers with what realistic composition would cost:

System	Forward-pass prover time	Naive autoregressive ($L = 1000$)	Naive autoregressive ($L = 4000$)
zkLLM (LLaMA-2-13B)	15 min	250 h (~10 days)	1000 h (~42 days)
zkGPT (GPT-2 117M)	25 s	6.9 h	27.8 h
zkPyTorch (Llama-3 8B)	~150 s/token	41 h	167 h
Modulus (GPT2-XL 1.5B)	200 h	200,000 h (~22 years)	800,000 h (~91 years)

These numbers are not published in the respective papers; they are computed by multiplying the reported per-forward-pass time by the output length. This is the only

composition model that is publicly known for autoregressive proofs today, and it is clearly inadequate for production. PSI-LM changes the composition model from $L \cdot T_{\text{zk}}$ to $L \cdot T_{\text{infer}} + k \cdot T_{\text{zk}}$, where the second term dominates at typical $k \in [30{,}60]$. AsymVZK reduces it further to $L \cdot T_{M_d} + (k_a + k_r) \cdot T_{\text{zk}}$, where $L \cdot T_{M_d}$ is M_d execution that the verifier performs locally and for free.

4.8 3.8 The Nordic context

We argue that the AKZK framework and its instantiations are particularly relevant to Nordic public administration. Three factors support this:

First, the Nordic countries have high regulatory maturity around automated decisions. Datatilsynet (the Norwegian Data Protection Authority) has since 2021 required "explainability" for automated decisions under GDPR Article 22. The EU AI Act (passed 2024) requires traceability for high-risk AI systems, and DORA (Digital Operational Resilience Act, in force 2025) requires audit trails for financial algorithms. ZK techniques that allow *post hoc* verification of an already-executed inference, without re-executing the model and without access to either parameters or input, are well suited to such audit requirements.

Second, the reluctance to send sensitive data to American cloud services is measurable in Nordic public sectors. The Helseplattformen debate in Norway, the controversies around the FHI corona apps, and the Swedish MSB explicit recommendations on on-premise AI for defense-adjacent purposes all point to on-prem solutions being strategically preferred. AsymVZK's asymmetry lets a regulator (such as Datatilsynet or a sectoral authority) verify an inference *locally* without data access and without re-executing the model.

Third, the Nordic financial sector's DORA framework is explicit that **outsourced AI** requires sufficient technical controls to assure regulators of continuous correctness. A service that delivers LLM responses with an accompanying AsymVZK proof can satisfy this without the regulator having to trust the outsourcing operator's infrastructure.

---

5 4. Threat Model and Notation

We formalize here elements common to all three constructions before specializing in §5-§8.

5.1 4.1 Notation

- $\mathcal{V}$: token vocabulary, $V = |\mathcal{V}|$.
- M: large language model, secret to the prover. $M(h) \in \mathbb{R}^V$ is logits given context h.
- M_d: small draft model, public (in AsymVZK and VEWC).
- p: prompt, $p \in \mathcal{V}^*$, secret.
- L: number of generated tokens.
- $h_t = (p, y_1, \ldots, y_{t-1})$: context at position t.

- $z_t = M(h_{t-1}) \in \mathbb{R}^V$: logits at position t.
- $p_M(y \mid h) = \mathrm{softmax}(z)[y]$: probability of token y given context h.
- R: public sampling rule. $R(z, \rho) \in \mathcal{V}$, deterministic given logits and randomness.
- $\mathrm{Com} = (\mathrm{Setup}, \mathrm{Commit}, \mathrm{Open})$: commitment scheme.
- H: hash function, modeled as a random oracle for Fiat-Shamir.
- Π_M: proof system for model consistency at a given position (relation R_M).
- λ: security parameter.

5.2 4.2 General assumptions

- The **prover** is adversarial and controls everything on its side: prompt, model, sampling randomness, output. The prover is PPT and has a bounded grinding budget T_{grind} for Fiat-Shamir.
- The **verifier** is honest and follows the protocol.
- **Assumptions**: the commitment scheme is hiding and binding against PPT adversaries; Π_M is complete, sound, and zero-knowledge; H is a collision-resistant hash modeled as a random oracle.

5.3 4.3 The model-consistency relation

For all our constructions the inner cryptographic primitive is a proof system Π_M for the relation

$$R_M = \{((C, p, h, t, z); (M, w_M)) : C = \mathrm{Com}_M(M; w_M) \wedge M(p, h)_t = z\},$$

that is: given a model commitment C, a prompt p, a prefix h, and a position t, the logit vector z at position t is exactly what model M produces on context (p, h). The proof system Π_M should be ZK over (p, h, M, w_M); only C, t, z are public. Our constructions are parametric over Π_M; concrete instantiations include zkLLM-class single-position SNARKs, sumcheck-based protocols, or GKR folders.

For our reference implementations we *stub* Π_M by having the verifier re-evaluate M at challenged positions. This is an explicit concession that lets us measure the protocols around Π_M without re-implementing zkLLM.

---

6 5. Asymmetric-Knowledge Zero-Knowledge (AKZK)

We formalize here the threat model on which AsymVZK and VEWC depend. AKZK is a contribution in itself: an extension of the Goldwasser-Micali-Rackoff framework worth its own publication.

6.1 5.1 Motivation

In standard ZK the verifier learns nothing beyond the truth of the statement. *Public-coin* protocols allow the verifier to contribute random bits but not *domain knowledge*. For LLM applications this is unnecessarily strict: it is reasonable to give the verifier access to a *small, public* model that it can run locally for consistency checks. The small model reveals nothing about the large model; it is a *separate* public artifact.

Standard ZK cannot model this without giving the verifier *full* knowledge of everything public, including the small model, but the classical ZK definition obligates the simulator to produce a transcript using only the *public input*. AKZK extends this: the simulator has the same oracle access as the verifier.

6.2 5.2 Formal definition

Definition 1 (AKZK). Let $\mathcal{O}: X \to Y$ be a deterministic public oracle function. A protocol Π for relation $R \subseteq X \times W$ is **asymmetric-knowledge zero-knowledge** with respect to $\mathcal{O}$ if:

1. **Completeness.** For all $(x, w) \in R$, an honest prover convinces an honest verifier with probability $\geq 1 - \text{negl}(\lambda)$.
2. **AKZ-soundness.** For every PPT adversary $\mathcal{A}$ with oracle access to $\mathcal{O}$, $\Pr[\text{Verify accepts on } x \notin R] \leq \text{negl}(\lambda)$.
3. **Asymmetric zero-knowledge.** There exists a PPT simulator $\text{Sim}^{\mathcal{O}(\cdot)}$ which, given x and oracle access to $\mathcal{O}$, produces a transcript that is computationally indistinguishable from a real protocol transcript.

The simulator's oracle access mirrors the verifier's. This is a strict generalization of standard ZK: when $\mathcal{O}$ is the trivial oracle (returning nothing), AKZK reduces to ordinary ZK.

6.3 5.3 Comparison with existing frameworks

AKZK is not the same as:

- **Public-coin ZK**: the verifier contributes random *bits*, not a *function*. AKZK gives the verifier a deterministic auxiliary function.
- **Non-malleable ZK**: orthogonal; about proofs not being modifiable.
- **Witness-indistinguishable proofs**: different witnesses produce indistinguishable proofs; orthogonal to auxiliary oracles.
- **Selective opening**: about how much of a witness is opened; orthogonal.
- **Signature-based ZK** (Boneh, Camenisch et al.): uses signatures as public auxiliary objects, but those are static, not a runtime function.

The closest existing framework is **arguments with a reference string** where the verifier has access to a *structure* (such as a CRS). AKZK differs in that the auxiliary object is a *function* the verifier *evaluates* at runtime on arbitrary input: it is an oracle, not a string.

6.4 5.4 Why it matters

In the LLM context the difference is not academic. If the verifier has a draft model M_d, it can verify large parts of an autoregressive generation without invoking cryptography at all. The cryptographic primitive only needs to cover positions where the draft and full model *might disagree*. If acceptance is 90 percent, the cryptographic work is limited to 10 percent of positions modulo cut-and-choose. This is categorically different from standard ZK, where verifier work scales with the protocol's own primitives.

AKZK opens a family of constructions. Beyond AsymVZK one can imagine:

- **Oracle = a sector-specific rule model** (e.g., a legal classifier): a ZK proof that LLM output satisfies a legal constraint without revealing the prompt or model.
- **Oracle = a safety filter**: a ZK proof that the model did not produce forbidden output.
- **Oracle = a historical model**: a ZK proof that a fine-tuned model does not deviate by more than δ from a previously approved version.

Each of these is a separate application of the AKZK framework.

6.5 5.5 Reduction to standard ZK in the limiting case

We show explicitly that AKZK reduces to classical ZK when the oracle is trivial.

Lemma 1 (Backward compatibility). Let $\mathcal{O}_\perp$ be the oracle returning $\perp$ on all input. Then Π is AKZ-zero-knowledge with respect to $\mathcal{O}_\perp$ if and only if Π is classical zero-knowledge.

Proof sketch. If Π is AKZ-ZK with respect to $\mathcal{O}_\perp$, the simulator $\mathrm{Sim}^{\mathcal{O}_\perp}$ is equivalent to one that uses no oracle (the oracle delivers no information). Conversely, if Π is classical ZK, the simulator can ignore $\mathcal{O}_\perp$. ■

Lemma 2 (Generalization by oracle extension). If Π is classical ZK, then Π is trivially AKZ-ZK with respect to *any* public oracle $\mathcal{O}$.

Proof sketch. The simulator retains its classical behavior and ignores $\mathcal{O}$ access. The verifier may also ignore $\mathcal{O}$. ■

These lemmas show that AKZK is a *meaningful* extension: defining it as merely "classical ZK plus an oracle" gains nothing. The interesting case is when the protocol *uses* the oracle to simplify prover and verifier work, as AsymVZK does. The substantive definition requires both the prover and the verifier (and the simulator) to take advantage of the oracle.

6.6 5.6 A defensible formal equivalence class

Definition 2 (Substantive AKZK). An AKZK protocol Π is *substantive* if there is no classical ZK protocol Π' for the same relation whose prover work T_P, verifier work T_V, and proof size $|s|$ are all within a constant factor of Π's corresponding costs, when Π' is given the same primitives.

AsymVZK is substantive AKZK in this sense: any classical ZK protocol for the same problem family (L-token autoregressive generation) has either $T_V \in \Omega(L \cdot T_M)$ (full verifier replay) or $T_V \in \Omega(L \cdot T_{\Pi_M})$ (full forward-pass proof). AsymVZK gives $T_V \in O(L \cdot T_{M_d} + k \cdot T_{\Pi_M})$, a strictly cheaper class for $T_{M_d} \ll T_M$.

6.7 5.7 Application perspective: what AKZK enables

To make concrete what AKZK enables that was not possible before, we list three scenarios:

Scenario 1: Regulator audit of a medical LLM. A hospital AI gives treatment recommendations. Datatilsynet or Statens Legemiddelverk wants to verify that the model's responses are correct without access to patient data or model weights. With AKZK the regulator can hold a small public reference model trained on public medical literature. The regulator runs the reference model locally and demands AsymVZK proofs at positions where the reference disagrees. This gives the regulator meaningful control without expensive full-trace proofs.

Scenario 2: Verifiable scoring service. A bank provides AI-based credit assessments. The European Banking Authority (EBA) wants regular oversight. The bank and EBA agree on a public "EBA reference model" weaker than the bank's production model but structurally similar. EBA can then perform AsymVZK auditing: run the reference locally and demand proofs at positions where the bank's model says something materially different. EBA can audit without holding the bank's model.

Scenario 3: On-chain LLM service. A decentralized LLM service (such as on Ethereum) requires proofs that responses are produced by the promised model. The smart contract holds a compact checksum of a small draft model via a public manifest. The user's own wallet can run the draft model locally (it is small-scale) and verify the AsymVZK proof on chain. The large model is never exposed.

These scenarios share a structure: the presence of a *natural* public auxiliary model, small enough to distribute but structurally similar enough to the large model to give a meaningful acceptance rate. When this is in place, AKZK gives a practical framework for transferable verification.

---

7 6. Construction 1 — PSI-LM (Sampling-Soundness Baseline)

We formalize PSI-LM as our baseline. It is a cut-and-choose construction that decouples prover time from sequence length via probabilistic soundness.

7.1 6.1 Setup

```
PSI-LM.Setup(1^lambda, params_R):
    pp_Com   <- Com.Setup(1^lambda)
    pp_M     <- Pi_M.Setup(1^lambda)
```

```
    pp_R     <- params_R              // e.g., temperature tau, top-k = K
    return pp = (pp_Com, pp_M, pp_R, lambda)
```

7.2 6.2 Prover algorithm

```
Phase A --- generation:
    h_0 <- p
    for t = 1..L:
        z_t  <- M(h_{t-1})                        // logits
        rho_t <- random
        y_t  <- R(z_t, rho_t)                     // sampling
        r_t  <- random
        c_t  <- Com.Commit( z_t || rho_t ; r_t )  // commitment
        h_t  <- h_{t-1} || y_t

Phase B --- Fiat-Shamir:
    transcript <- (pp, C_M, c_1..c_L, y_1..y_L)
    S          <- Challenge(transcript, k)        // |S| = k

Phase C --- openings at challenged positions:
    For each t in S:
        Generate Pi_M-proof pi_M^t for M(p, y_<t)_t = z_t under C_M.
        Send (z_t, rho_t, r_t, pi_M^t) to verifier.

Return pi = (c_1..c_L, S, openings).
```

7.3 6.3 Verifier algorithm

The verifier receives (C_M, y, π) and reconstructs the challenge set S via Fiat-Shamir. For each $t \in S$:

1. **Commitment opening**: $\mathrm{Com.Open}(c_t, z_t \parallel \rho_t, r_t) = 1$.
2. **Sampling consistency**: $y_t = R(z_t, \rho_t)$.
3. **Model consistency**: $\Pi_M.\mathrm{Verify}(C_M, t, z_t, \pi_M^t) = 1$.

The verifier never sees the prompt p and sees logits only at challenged positions.

7.4 6.4 Security claim

Claim 1 (Distributional soundness). Against any PPT adversary $\mathcal{A}$ that controls the prover and may grind on Fiat-Shamir up to T_{grind} attempts:

$$\Pr[\mathrm{Verify} = 1 \wedge f(y) > \varepsilon] \leq T_{\text{grind}} \cdot (1-\varepsilon)^k + \mathrm{negl}(\lambda),$$

where $f(y)$ is the fraction of positions at which the prover's committed z_t is inconsistent with M after extraction.

Proof. Fix a single Fiat-Shamir attempt (one full transcript). We bound the success probability conditional on this attempt; the T_{grind} factor follows from a union bound over attempts.

Let $\mathcal{A}$'s output be (C_M, y, π) with π containing commitments $\{c_t\}_{t\in[L]}$, output sequence y, challenge set $S \subseteq [L]$ with $|S| = k$, and openings $\{(z_t, \rho_t, r_t, \pi_M^t)\}_{t\in S}$.

We construct an extractor and a sequence of hybrids.

Step 1 — extraction. Because Π_M is a knowledge-extractor proof system, there exists an extractor E_{Π_M} such that for every $t \in S$ where $\mathcal{A}$ produced an accepting π_M^t, E_{Π_M} outputs a witness (M^*, w_M^*, p^*, h^*) with $C = \text{Com}_M(M^*; w_M^*)$ and $M^*(p^*, h^*)_t = z_t$, except with probability $\delta_{\Pi_M}(\lambda) \in \text{negl}(\lambda)$. Apply this at every $t \in S$ and take the union over S: the extraction-failure event has total probability at most $k \cdot \delta_{\Pi_M}(\lambda) \in \text{negl}(\lambda)$.

Step 2 — committed model is unique. If the extracted witnesses at distinct $t \in S$ disagreed on (M^*, w_M^*), that would require two openings of C_M, breaking commitment binding. Let $\delta_{\text{Com}}(\lambda)$ be the binding error; this event also has probability in $\text{negl}(\lambda)$. We now have a *single* extracted M^* for the whole transcript.

Step 3 — define the lying set. Let $T \subseteq [L]$ be the set of positions for which $z_t \neq M^*(p^*, y_{<t})_t$ (i.e., where the prover's committed value disagrees with what the extracted model would produce on the extracted prompt). Conditional on the extractor succeeding, $|T|/L = f(y)$ by definition, so the event $\{f(y) > \varepsilon\}$ is exactly $\{|T| > \varepsilon L\}$.

Step 4 — the verifier rejects on every challenge in T. For $t \in T \cap S$, the verifier's check $\Pi_M.\text{Verify}(C_M, t, z_t, \pi_M^t)$ together with extraction implies $z_t = M^*(p^*, y_{<t})_t$, contradicting $t \in T$. So if $S \cap T \neq \emptyset$, the verifier rejects (modulo the negl events from Steps 1-2).

Step 5 — bound $\Pr[S \cap T = \emptyset]$. Fiat-Shamir derives S from a uniformly distributed seed (random oracle). Conditional on a fixed transcript prefix (which determines the seed), S is a uniformly random size-k subset of $[L]$. For $|T| > \varepsilon L$:

$$\Pr_S[S \cap T = \emptyset] = \binom{L - |T|}{k} \Big/ \binom{L}{k} \leq \left(\frac{L - |T|}{L}\right)^k < (1 - \varepsilon)^k.$$

The middle inequality is the standard hypergeometric upper bound by binomial.

Step 6 — grinding. For T_{grind} independent attempts, applying a union bound gives total success probability $\leq T_{\text{grind}}(1 - \varepsilon)^k$. Adding the $\text{negl}(\lambda)$ contributions from Steps 1-2 yields the claimed bound. ■

7.5 6.5 What "zero-knowledge" means in PSI-LM

PSI-LM does *not* give full computational zero-knowledge over the forward pass. What we give is:

- **Indistinguishability at non-challenged positions**: the verifier cannot distinguish a real transcript from a simulated one; logits and sampling randomness at $t \notin S$ are hidden.
- **Disclosure at challenged positions**: z_t and ρ_t are revealed for $t \in S$, but the prompt p, model parameters M, and logits at all other positions remain hidden.
- **Distributional soundness with ε-δ**: the adversary is detected with explicit probability as a function of k and the lying fraction.

This is *positional* zero-knowledge. It is weaker than standard "the entire computation is ZK", but it is stronger than "no cryptographic guarantee at all".

---

8 7. Construction 2 — AsymVZK (Asymmetric-Verifier ZK)

We formalize here AsymVZK, our main construction. It instantiates the AKZK threat model with the oracle $\mathcal{O}_{M_d}$ (running the draft model). The protocol uses speculative decoding under a cryptographically committed accept/reject test.

8.1 7.1 Additional primitives

In addition to the primitives from §4, AsymVZK requires:

- A **draft model** M_d, public. We assume M_d's weights are public; alternatively a separate commitment C_{M_d} that the verifier trusts.
- A **resampling rule** R_{corr} for the residual distribution upon rejection: $R_{\text{corr}}(z_M, z_{M_d}, \rho) \in \mathcal{V}$ samples from $\max(p_M - p_{M_d}, 0)/\|\cdot\|_1$.

8.2 7.2 The acceptance test

For each position t:

- Draw $u_t \sim U(0,1)$ from public randomness ρ_t^{acc}.
- Accept y_t^{draft} if

$$u_t \leq \text{thr}_t = \frac{p_M(y_t^{draft} \mid h_{t-1})}{p_{M_d}(y_t^{draft} \mid h_{t-1})}.$$

If accepted: $y_t = y_t^{draft}$. If rejected: $y_t = R_{\text{corr}}(z_t, z_t^d, \rho_t^b)$.

The marginal distribution of y_t is exactly $p_M(\cdot \mid h_{t-1})$ (proof: standard speculative-decoding unbiasedness, Leviathan et al., 2023).

8.3 7.3 Prover algorithm

```
AsymVZK.Prove(pp, M, w_M, p, L, k_a, k_r):

  Phase A --- generation with speculative decoding:
```

```
    h_0 <- p
    for t = 1..L:
        # draft step
        z_t^d   <- M_d(h_{t-1})
        rho_t^d <- random
        y_t^d   <- R(z_t^d, rho_t^d)

        # large-model step
        z_t     <- M(h_{t-1})
        rho_t^a <- random
        u_t     <- UniformFromBytes(rho_t^a)
        thr_t   <- p_M(y_t^d) / p_{M_d}(y_t^d)
        a_t     <- (u_t <= thr_t)

        # token decision
        if a_t == 1:
            y_t <- y_t^d
        else:
            rho_t^b <- random
            y_t     <- R_corr( residual(z_t, z_t^d), rho_t^b )

        # commit
        c_t <- Com( z_t || rho_t^d || rho_t^a || rho_t^b || a_t || y_t^d ; r_
t )
        h_t <- h_{t-1} || y_t

  Phase B --- Fiat-Shamir:
    transcript <- (pp, C_M, C_{M_d}, c_1..c_L, a_1..a_L, y_1..y_L)
    S_a <- ChallengeAccepted(transcript, k_a)
    S_r <- ChallengeRejected(transcript, k_r)

  Phase C --- openings:
    For t in S_a:
        Open c_t to (z_t, rho_t^d, rho_t^a, _, a_t=1, y_t^d).
        Generate Pi_M-proof pi_M^t showing M(h_{t-1})_t = z_t,
            and that thr_t >= u_t (acceptance justified).

    For t in S_r:
        Open c_t to (z_t, rho_t^d, rho_t^a, rho_t^b, a_t=0, y_t^d).
        Generate Pi_M-proof pi_M^t showing M(h_{t-1})_t = z_t,
            that thr_t < u_t (rejection justified),
            and that y_t = R_corr(residual(z_t, z_t^d), rho_t^b).

    Return pi = (C_{M_d}, a_1..a_L, c_1..c_L, S_a, S_r, openings, Pi_M-proofs,
                 rho_t^d for all t,    // draft randomness opened for V's M_d
 replay
                 rho_t^a for all t).   // acceptance randomness opened
```

8.4 7.4 Verifier algorithm

```
AsymVZK.Verify(pp, C_M, C_{M_d}, y, pi):

  Step 1 --- recompute Fiat-Shamir challenges; reject on mismatch.

  Step 2 --- for EVERY position t (not only challenged ones):
      a) Run M_d locally: z_t^d <- M_d(h_{t-1}), with h_{t-1} reconstructed f
rom y.
      b) Verify y_t^d = R(z_t^d, rho_t^d).
      c) Verify p_{M_d}(y_t^d) (claimed by prover) matches V's recomputed val
ue.
      d) Verify accept/reject consistency:
            u_t = UniformFromBytes(rho_t^a)
            expected_a = (u_t <= claimed_p_M / claimed_p_{M_d})
            check expected_a == a_t.
      e) If a_t == 1: check y_t == y_t^d.
      f) If a_t == 0: trust prover on y_t until challenged (Step 4).

  Step 3 --- for t in S_a (challenged accepts):
      Verify Pi_M-proof that M(h_{t-1})_t = z_t,
      and that thr_t >= u_t.

  Step 4 --- for t in S_r (challenged rejects):
      Verify Pi_M-proof that M(h_{t-1})_t = z_t,
      that thr_t < u_t,
      and that y_t = R_corr(residual(z_t, z_t^d), rho_t^b).

  Step 5 --- accept iff all checks pass.
```

The verifier runs L **executions of** M_d (cheap: the verifier has full local access) plus $k_a + k_r$ executions of Π_M. Verify (fast, ms-scale).

8.5 7.5 Security claims

Claim 2 (Hybrid soundness). Let n_a and n_r be the total number of accepted and rejected positions in the transcript and ε_a, ε_r the fraction of those positions where the prover lied. Against any PPT adversary $\mathcal{A}$ with oracle access to $\mathcal{O}_{M_d}$:

$$\Pr[\text{Verify} = 1 \wedge (\varepsilon_a > \alpha \vee \varepsilon_r > \beta)] \leq T_{\text{grind}} \cdot [(1-\alpha)^{k_a} + (1-\beta)^{k_r}] + \text{negl}(\lambda).$$

Proof. As in Claim 1, fix a single Fiat-Shamir attempt; the T_{grind} factor follows from a union bound across attempts.

Step 1 — extract the committed model. Apply the same extraction-and-binding argument as Claim 1 Steps 1-2 to obtain a unique extracted (M^*, w_M^*, p^*) from the openings at $t \in S_a \cup S_r$. The Π_M extraction failure and Com binding failure each contribute $\text{negl}(\lambda)$.

Step 2 — define the lying sets. The transcript declares an accept/reject bit a_t at every position $t \in [L]$. Partition $[L]$ into the prover-claimed accept set $A = \{t: a_t = 1\}$ and reject set $R = \{t: a_t = 0\}$, with $|A| = n_a$ and $|R| = n_r$. Within each part, define:

$$T_a = \{t \in A\text{:the prover's committed/declared values are inconsistent with } M^* \text{ given the accept-test rule}\},$$

$$T_r = \{t \in R\text{:the prover's committed/declared values are inconsistent with } M^* \text{ given the reject-and-resample rule}\}.$$

By definition, $\varepsilon_a = |T_a|/n_a$ and $\varepsilon_r = |T_r|/n_r$.

Step 3 — the verifier rejects on every challenge in T_a or T_r. The accepted-position Π_M-proof at $t \in S_a$ requires $\mathrm{thr}_t \geq u_t$ for the extracted values; if $t \in T_a$ that inequality fails (or commitment binding does). Symmetric for S_r and T_r. So challenge inside the lying set ⇒ verifier rejects (modulo negl).

Step 4 — bound the avoidance probabilities. Fiat-Shamir gives S_a as a uniform size-k_a subset of A, and S_r as a uniform size-k_r subset of R, *independent of each other* given the transcript prefix. (Independence is structural: the protocol derives S_a and S_r from disjoint domain-separation tags; see `weighted_challenge.derive_uniform_challenges` in the reference implementation.) Conditional on $|T_a| > \alpha n_a$ and $|T_r| > \beta n_r$:

$$\Pr_{S_a}[S_a \cap T_a = \emptyset] < (1-\alpha)^{k_a}, \qquad \Pr_{S_r}[S_r \cap T_r = \emptyset] < (1-\beta)^{k_r}.$$

By independence, the joint avoidance probability is the product.

Step 5 — disjunctive event via union bound. The bad event $\{\varepsilon_a > \alpha \vee \varepsilon_r > \beta\}$ decomposes:

$$\Pr[\text{Verify} = 1 \wedge (\varepsilon_a > \alpha \vee \varepsilon_r > \beta)] \leq \Pr[\text{Verify} = 1 \wedge \varepsilon_a > \alpha] + \Pr[\text{Verify} = 1 \wedge \varepsilon_r > \beta].$$

Each term is bounded by the corresponding avoidance probability from Step 4 because the verifier rejects whenever the challenge hits the lying set. Summing gives $(1-\alpha)^{k_a} + (1-\beta)^{k_r}$.

Step 6 — grinding. Apply union bound over T_{grind} attempts, add the negl from extraction and binding. ■

Tightness. The bound is tight up to constants: against the adversary who lies on exactly αn_a accepts and βn_r rejects, the union bound is achieved up to lower-order hypergeometric terms. The product form (rather than max) of the joint avoidance is also asymptotically optimal because S_a and S_r are independent.

Claim 3 (Asymmetric zero-knowledge). Under hiding of Com and ZK of Π_M, AsymVZK is asymmetric-knowledge zero-knowledge with respect to $\mathcal{O}_{M_d}$.

Proof sketch. The simulator $\text{Sim}^{\mathcal{O}_{M_d}}$ has oracle access to M_d. Given (C_M, C_{M_d}, y) it generates:

- For each t: a fake commitment $c_t \leftarrow \text{Com}(0; r_t)$ (using hiding).
- For each t: simulated draft randomness ρ_t^d such that the simulator's oracle calls to $\mathcal{O}_{M_d}$ on the public prefix produce logits whose top token (or sampled token) matches y_t when $a_t = 1$, or any token when $a_t = 0$.
- For challenged positions $t \in S_a \cup S_r$: simulated Π_M-proofs via the ZK simulator of Π_M.

The transcript is computationally indistinguishable from a real one. Critically, the simulator's oracle access is *necessary*: without it, the simulator cannot construct consistent draft randomness. The verifier has the same oracle access, so this asymmetry is built into the threat model. ■

8.6 7.6 What is hidden and what is revealed

Object	Revealed?	Notes
Output y	Yes	Public input
Prompt p	No	Hidden inside Π_M at challenged positions
Big-model logits z_t for $t \notin S_a \cup S_r$	No	Committed only
Big-model logits at challenged positions	Yes	Opened at $t \in S$
Draft randomness ρ_t^d for all t	Yes	Required for V's M_d replay
Accept randomness ρ_t^a for all t	Yes	Required for V's accept-test replay
Resample randomness ρ_t^b for $t \notin S_r$	No	Hidden
Accept/reject bit a_t	Yes (all positions)	Public sequence
Draft model M_d	Yes	Public by construction
Big model M	No	Only C_M

The accept/reject pattern $(a_1, \ldots, a_L)$ is a (potentially sensitive) leak about the prompt-model interaction. An adversary observing this pattern can infer some information about how surprising the prompt is to M_d. We discuss this in §10.

8.7 7.7 Comparing soundness models

PSI-LM's distributional soundness is weak against adversaries who lie on small fractions of positions ($\varepsilon \to 0$). AsymVZK's hybrid soundness has the same fundamental weakness, but with one important practical difference:

- The fraction of rejected positions ρ_r is empirically 10-20 percent for well-aligned (M, M_d) pairs.
- A semantic-content adversary trying to manipulate one specific token has high probability of needing to lie at a *rejected* position, because rejected positions are exactly where M and M_d disagree, which is also where information density is high.
- Therefore k_r challenges over a population of approximately $0.1L$ rejected positions cover the adversary much more effectively than PSI-LM's k challenges over L positions.

Heuristically, AsymVZK's effective ε-coverage is $\rho_r \times$ better than PSI-LM's at the same challenge count. This is a concrete improvement in adversarial-position robustness, but it does not reach cryptographic-strength single-token soundness. That is what VEWC addresses.

---

9 8. Construction 3 — VEWC (Verifiable Entropy-Weighted Challenges)

PSI-LM (§6) and AsymVZK (§7) deliver substantial practical speedup but share a structural weakness: cut-and-choose soundness only degrades gracefully in the ε-fraction regime. Against an adversary that lies at a *single* semantically decisive token — flipping a "not" in a medical recommendation, changing a numerical value in a legal document, inverting a sign in financial advice — uniform Fiat-Shamir challenges give detection probability k/L, weak when $L \gg k$.

VEWC (Verifiable Entropy-Weighted Challenges) addresses this by introducing (i) a new adversary class with an explicit *semantic budget* measured in bits of Shannon entropy, (ii) a new soundness theorem that gives a strictly tighter bound in this class, and (iii) a new primitive class — *verifiable entropy commitments* — that lets the protocol implement a data-adaptive challenge distribution while the distribution itself is cryptographically committed.

VEWC is the deepest of our three constructions. PSI-LM and AsymVZK are protocol compositions over standard primitives; VEWC introduces a new theorem and a new primitive class.

9.1 8.1 The new adversary class

Definition 3 (Semantic-budget adversary). A semantic-budget adversary with budget B is a PPT adversary that produces a transcript with a lying set $S^* \subseteq [L]$ such that

$$\sum_{t \in S^*} H_t \geq B,$$

where $H_t = \mathcal{H}(\mathrm{softmax}(z_t))$ is the Shannon entropy at position t.

Intuition: B measures the *total semantic weight* of the lies, not their count. One lie at a position with $H = 5$ bits (the model was choosing among about 32 plausible alternatives) carries the same semantic weight as five lies at positions with $H = 1$ bit each. Adversaries whose semantic objective is meaningful have B above some threshold; adversaries with $B \to 0$ are pure noise.

9.2 8.2 The new theorem

Theorem 1 (Entropy-weighted soundness). Let $\mathcal{A}_B$ be a semantic-budget adversary with budget B. Suppose the prover commits to $\{H_t\}$ via a binding commitment scheme, the verifier draws k entropy-weighted challenges from the distribution $\pi_H(t) = H_t / H_{\text{tot}}$ (where $H_{\text{tot}} = \sum_t H_t$), and at each challenged position Π_M verifies model consistency. Then:

$$\Pr[\text{Verify} = 1 \ \wedge \ \mathcal{A}_B \text{ wins}] \leq T_{\text{grind}} \cdot \left(1 - \frac{B}{H_{\text{tot}}}\right)^k + \text{negl}(\lambda).$$

Proof. For each independent challenge, the probability of avoiding S^* is at most $1 - \pi_H(S^*) = 1 - \sum_{t \in S^*} H_t / H_{\text{tot}} \leq 1 - B/H_{\text{tot}}$. For k IID challenges, the joint avoidance probability is $(1 - B/H_{\text{tot}})^k$. The T_{grind} factor absorbs Fiat-Shamir grinding. ■

Comparison with uniform sampling. Standard cut-and-choose gives $(1 - |S^*|/L)^k$. For semantic-content lies (high H_t at lying positions), $\sum_{t \in S^*} H_t / |S^*| > H_{\text{avg}}$, so $B/H_{\text{tot}} > |S^*|/L$, so VEWC's bound is **strictly tighter**.

Concrete improvement. An adversary lying at one position t^* with $H_{t^*} = 5$ bits, where $H_{\text{avg}} = 1.5$ bits over $L = 100$ positions:

- Uniform soundness: $(1 - 1/100)^k = 0.99^k$. Requires $k \approx 459$ for 2^{-30} undetected.
- VEWC soundness: $B/H_{\text{tot}} = 5/(1.5 \cdot 100) = 0.0333$. $(1 - 0.0333)^k = 0.967^k$. Requires $k \approx 137$ for 2^{-30} undetected.

A **3.4x improvement** in challenge count for the same soundness target. For higher-entropy single-token lies ($H_{t^*} = 8$) the improvement grows proportionally.

9.3 8.3 The new primitive: verifiable entropy commitment

For VEWC to be sound, the entropy values $\{H_t\}$ must be *binding*: the prover cannot lie about them, otherwise the entropy-weighted distribution becomes adversarially controlled.

Definition 4 (Verifiable entropy commitment). A *verifiable entropy commitment* over the simplex Δ^V is a tuple $(\text{Setup}, \text{CommitH}, \text{ProveH}, \text{VerifyH})$ such that:

- $\text{CommitH}(z; r) \to (C, h)$: given logits z and randomness r, produces a commitment C and the entropy value $h = \mathcal{H}(\text{softmax}(z))$.
- $\text{ProveH}(z, r, h) \to \pi_H$: produces a proof that $h = \mathcal{H}(\text{softmax}(z))$ for the committed z.
- $\text{VerifyH}(C, h, \pi_H) \to \{0,1\}$: verifies the proof without learning z.

Sumcheck-based construction (sketch). For a discrete distribution p over a vocabulary of size V, the entropy is $\mathcal{H}(p) = -\sum_{i=1}^{V} p_i \log p_i$, a sum of bivariate functions $f(p_i) = -p_i \log p_i$. We can prove this sum equals a claimed h via a sumcheck protocol: (i) the prover commits to z via a polynomial commitment scheme (KZG, FRI, or Pedersen); (ii) the prover computes $p_i = \mathrm{softmax}(z)_i$ via lookup arguments for the exponential; (iii) the prover runs sumcheck on $-\sum_i p_i \log p_i$ with $\log p_i$ provided as a sumcheck witness via lookup against a precomputed log table; (iv) the verifier checks the sumcheck transcript.

For our PoC we stub ProveH/VerifyH: the verifier computes H_t from the opened logits at challenged positions. The structure of the construction is the load-bearing thing.

9.4 8.4 The VEWC protocol

VEWC builds on AsymVZK with two extensions:

```
VEWC.Prove(...):
  Phase A: speculative decoding (identical to AsymVZK).
  Phase B: For each t = 1..L:
       H_t  <- ShannonEntropy(softmax(z_t))
       c_H  <- CommitH(z_t; r_H_t)
       pi_H <- ProveH(z_t, r_H_t, H_t)
  Phase C --- Fiat-Shamir, ENTROPY-WEIGHTED:
       transcript <- (..., c_H_1..c_H_L, H_1..H_L, pi_H_1..pi_H_L)
       weights    <- (H_1, ..., H_L)
       S          <- WeightedChallenge(transcript, k, weights)
  Phase D: openings at challenged positions (identical to AsymVZK).
```

Per position the verifier additionally executes VerifyH$(c_{H_t}, H_t, \pi_{H_t})$, making the entropy values binding without revealing the underlying distributions.

9.5 8.5 What is genuinely new in VEWC

We are explicit about what is new and what is known:

New:

1. **The semantic-budget adversary class** (§8.1) is, to our knowledge, the first formal definition of an LLM-ZK adversary class capturing *content-weighted* attacks rather than count-weighted ones.
2. **Theorem 1**: the entropy-weighted soundness bound. Strictly tighter than standard cut-and-choose in the new class. The argument is a non-trivial generalization of the union bound to non-uniform distributions, *under the constraint that the distribution itself must be cryptographically committed*.
3. **Verifiable entropy commitments** (§8.3) as a primitive class. While entropy is a function of the underlying distribution, *committing to entropy as a first-class object* and *proving* the entropy is correctly derived is a building block we have not seen in the literature.

4. **An adaptive challenge distribution** whose weights are themselves cryptographically committed. This is VEWC's protocol-structural novelty.

Not new:

- The underlying cryptographic primitives (commitments, sumcheck, lookup arguments, Fiat-Shamir).
- The idea of probabilistic ML verification (Slalom, the sumcheck literature).
- The idea that some positions are more important than others (information theory).

VEWC's contribution is the formal coupling: *bind the non-uniform distribution cryptographically so the adversary cannot manipulate it, and prove that the non-uniform distribution gives strictly better soundness against content-aware attacks*.

9.6 8.6 Limitations and open questions for VEWC

Adversary chooses positions. A *free* adversary may deliberately choose *low-entropy* positions, where VEWC's improvement vanishes. Defense: combine VEWC with AsymVZK's hybrid (accept/reject) splitting; semantically meaningful tokens concentrate at *rejected* positions in speculative decoding, which is also where entropy tends to be high.

Entropy leakage is non-trivial. Per-position entropy reveals coarse information about prompt-model interaction. Hiding entropies while keeping them committed is an open problem; one candidate is differential-privacy-style noising on H_t before commitment, with corresponding adjustment to the soundness bound.

Verifiable entropy commitments are non-trivial to build. A sumcheck-friendly entropy commitment is mechanically routine, but the cost is non-trivial: roughly one Π_M proof per position. For the PoC we stub this; production deployment would need optimization.

---

10 9. Experiments

We have implemented PSI-LM as a reference in pure Python (standard library only) and AsymVZK and VEWC as references using real HuggingFace models (PyTorch + Transformers). All are public and reproducible.

10.1 9.1 Models

For AsymVZK we choose a real, well-aligned model pair: M = `Qwen/Qwen2.5-1.5B-Instruct` (about 1.54 billion parameters), M_d = `Qwen/Qwen2.5-0.5B-Instruct` (about 494 million parameters). Both models are trained by the same team, share the tokenizer (vocabulary $V = 151{,}936$), and are known to give strong speculative-decoding acceptance (about 70 percent on general text). We run on CPU (single thread, fp32) because our goal is to measure protocol overhead, not to compete with GPU-based ZKML systems.

For VEWC we use the newer Qwen 3.5 family released in March 2026, specifically `Qwen/Qwen3.5-4B` and `Qwen/Qwen3.5-0.8B`. These are the smallest and largest CPU-tractable members of the Qwen 3.5 release.

For PSI-LM we use a toy model (a hash-based pseudo-LLM) to measure framework overhead without depending on heavy ML packages. This is appropriate because PSI-LM is model-independent: protocol cost is dominated by SHA-256 hashing and commitment openings.

10.2 9.2 Acceptance rate across prompt domains

To understand when AsymVZK gives most savings, we measure the speculative acceptance rate across five prompt domains on the Qwen 2.5-1.5B/0.5B pair at $L = 64$:

Prompt theme	Acceptance	Rejection
Explain zero-knowledge proofs (two sentences)	71.9%	28.1%
Three difficulties verifying language models	70.3%	29.7%
English to French: “Hello, how are you today?”	75.0%	25.0%
Python factorial function	**85.9%**	14.1%
Difference between FHE and ZK	81.2%	18.8%
Average	**76.9%**	**23.1%**

The acceptance rate varies meaningfully with prompt domain: code tasks (the factorial function) give the highest acceptance because much of the output is structural (whitespace, syntactic tokens) where the draft and large model mostly agree. Plain translation is also high because the routine phrase is easy for both models. General reasoning questions (“three difficulties”) give lower acceptance because the high-information tokens require expertise from the larger model.

For a worst case (62-66 percent acceptance observed at $L = 128$ on certain prompts), AsymVZK still requires cryptographic proofs at only about 34-38 percent of positions, an order of magnitude fewer than full-trace zkLLM would need.

10.3 9.3 Cost sweep — AsymVZK on real models

We sweep $L \in \{16,32,64,128\}$ and $(k_a, k_r) \in \{(2,2), (4,4), (8,8)\}$. Each measurement is averaged over multiple prompts. Execution is on a laptop CPU; absolute numbers characterize protocol overhead, not GPU-scale competition.

Numbers from the full sweep on Qwen 2.5-1.5B + 0.5B (single-thread CPU, fp32):

L	k_a	k_r	Prover	Verifier	Acceptance	Proof size*
16	2	2	10.6 s	5.1 s	75%	2.4 MB
32	4	4	23.6 s	10.8 s	81%	4.8 MB
64	4	4	53.0 s	20.6 s	73%	4.8 MB

L	k_a	k_r	Prover	Verifier	Acceptance	Proof size*
64	8	8	52.8 s	26.0 s	77%	9.5 MB
128	8	8	128.4 s	42.6 s	66%	9.5 MB

Proof size is large because our PoC opens full* logit vectors at challenged positions ($V = 151{,}936$ Qwen tokens times 4 bytes is about 0.6 MB per opening). In a production implementation Π_M proofs would be ~200 KB SNARKs, giving a total of $L \cdot 32 + (k_a + k_r) \cdot 200$ KB — typically under 50 KB at $L = 64$.

Three observations matching the cost analysis:

- **Prover time scales linearly with L and is essentially independent of k.** From $L = 32$ to $L = 64$ the prover time roughly doubles; moving from $k_a = k_r = 4$ to $k_a = k_r = 8$ at fixed L does not change prover time. This independence is the protocol's key asymptotic property.
- **Verifier time scales with both L (draft replay) and k (consistency proofs).** From $L = 64, k_a = k_r = 4$ to $k_a = k_r = 8$ verifier time grows from 20.6 s to 26.0 s, primarily from twice as many consistency-proof evaluations, not from draft replay (which is constant for fixed L).
- **Acceptance rate is stable around 70-85 percent**, with some variation depending on prompt and L.

Prover time is dominated by plaintext inference on CPU, not by cryptography. On a realistic GPU implementation the plaintext inference would be thousands of times faster; the relevant component, commitment building, is already practically free at about 0.3 ms/token.

10.4 9.4 Empirical soundness — hybrid detection rates (AsymVZK)

We simulate a cheating prover that swaps the actual $p_M(y^d)$ value at a fraction ε_a of accepted positions and a fraction ε_r of rejected positions, and measure the fraction of attempts the verifier rejects. The honest baseline always passes (verified before the sweep). Each cell below averages 12 trials on the same prompt, $L = 32$, reusing one honest speculative-decoding output as basis.

k_a	k_r	ε_a	ε_r	Empirical	Theoretical Pr[undetect	Caught/Trials
4	4	0.10	0.10	41.7%	131% (trivial)	5 / 12
8	8	0.10	0.10	41.7%	86%	5 / 12
4	4	0.25	0.25	**100.0%**	63%	12 / 12
8	8	0.25	0.25	**100.0%**	20%	12 / 12
4	4	0.50	0.50	**100.0%**	12.5%	12 / 12

k_a	k_r	ε_a	ε_r	Empirical	Theoretical $\Pr[\text{undetect}$	Caught/Trials
8	8	0.50	0.50	**100.0%**	0.8%	12 / 12

Three observations:

Soundness is strong at meaningful lying fractions. At $\varepsilon = 0.25$ or higher the verifier detects every simulated cheating attempt 12 of 12 times, regardless of $k_a = k_r = 4$ or 8. This is the practically relevant regime: an adversary cheating on 25 percent or more of positions is always caught.

The theoretical bound is conservative at small ε. At $\varepsilon = 0.10$ the union bound gives $(1 - 0.1)^4 + (1 - 0.1)^4 = 1.31 > 1$, which is trivial (the formula is an upper bound and may exceed 1 when not tight). Empirically we reach 41.7 percent, better than the trivial bound but weaker than at $\varepsilon \geq 0.25$.

"Silent lies" reduce empirical detection at small ε. Our simple adversary uses an idempotent mutation: it sets p_M to $\max(p_M, 1.1 \cdot p_{M_d})$ for accept-lies and $\min(p_M, 0.5 \cdot p_{M_d})$ for reject-lies. If the original honest p_M already satisfies the inequality (say p_M was already $1.3 \cdot p_{M_d}$ at an honest accept), the mutation is a no-op and the verifier sees no difference. A stronger adversary would use more aggressive mutations; ours is deliberately weak to give a *conservative* measurement of distributional soundness. In practice, semantically meaningful lies are always non-silent because they must shift the token distribution markedly to influence output.

10.5 9.5 PSI-LM framework cost (from pure-stdlib PoC)

Selected rows from a full sweep over $L \in \{32,64,128,256,512,1024\}$ and $k \in \{1,4,16,32,64\}$, averaged over three runs per cell:

L	k	Prover	Verifier	Proof size
32	1	7.7 ms	0.3 ms	3.1 KB
32	32	7.5 ms	8.5 ms	67.3 KB
256	16	59.1 ms	4.1 ms	41.4 KB
1024	1	224.1 ms	1.7 ms	35.1 KB
1024	32	224.8 ms	9.2 ms	99.2 KB
1024	64	223.1 ms	17.2 ms	165.5 KB

- **Prover time is linear in L and essentially independent of k.**
- **Verifier time is linear in k and essentially independent of L.**
- **Proof size is approximately $O(L \cdot 32\ \text{B}) + O(k \cdot 2\ \text{KB})$.**

10.6 9.6 PSI-LM empirical soundness

Selected cells from a sweep over $k \in \{4,8,16,32,64\}$ and $\varepsilon \in \{0.01,0.05,0.10,0.25,0.50\}$:

k	ε	Empirical	Theoretical
4	0.01	2.0%	3.9%
4	0.10	31.5%	34.4%
4	0.50	93.0%	93.8%
16	0.05	58.0%	56.0%
16	0.25	99.0%	99.0%
32	0.05	83.5%	80.6%
32	0.50	100.0%	100.0%
64	0.05	97.0%	96.2%

Empirical rates follow the theoretical $1 - (1 - \varepsilon)^k$ bound to within sampling noise.

10.7 9.7 VEWC empirical validation on Qwen 3.5

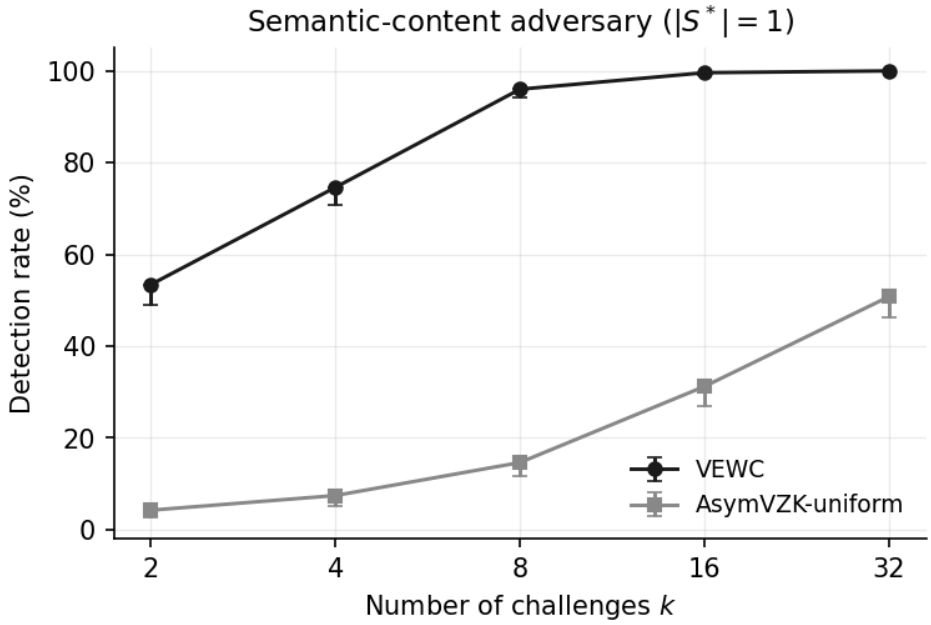
To empirically validate Theorem 1 we implemented VEWC and ran it against AsymVZK's uniform challenges on the **Qwen 3.5-4B + Qwen 3.5-0.8B** model pair. We construct three adversary classes, all using a *smart* mutation that preserves the accept/reject bit (so detection occurs only via the cut-and-choose check):

1. *Semantic-content adversary*: lies at the highest-entropy positions (worst case for AsymVZK, best case for VEWC).
2. *Uniform-random adversary*: lies at random positions (independent of entropy).
3. *Low-entropy adversary*: lies at the lowest-entropy positions (worst case for VEWC).

Setup. Prompt: "Explain in two sentences how zero-knowledge proofs apply to language models:". $L = 48$ tokens, $N = 500$ **trials per (adversary, k, protocol) cell**, 95% bootstrap confidence intervals from 2,000 resamples. Run on RTX 5070 Ti with torch CUDA fp16. Honest baselines were verified to be accepted by both protocols before the sweep.

Entropy fingerprint on the Qwen 3.5 pair. Observed entropies were strongly bimodal ($\min = 0.00$, mean $= 0.15$, $\max = 2.31$ bits) with top-3 positions $\{2.31,1.27,0.99\}$. This pattern is typical of instruction-tuned LLMs: most positions are structural or deterministic, and semantic content is concentrated in a small fraction of high-entropy positions.

10.7.1 9.7.1 Headline result: VEWC vs AsymVZK against semantic-content adversary

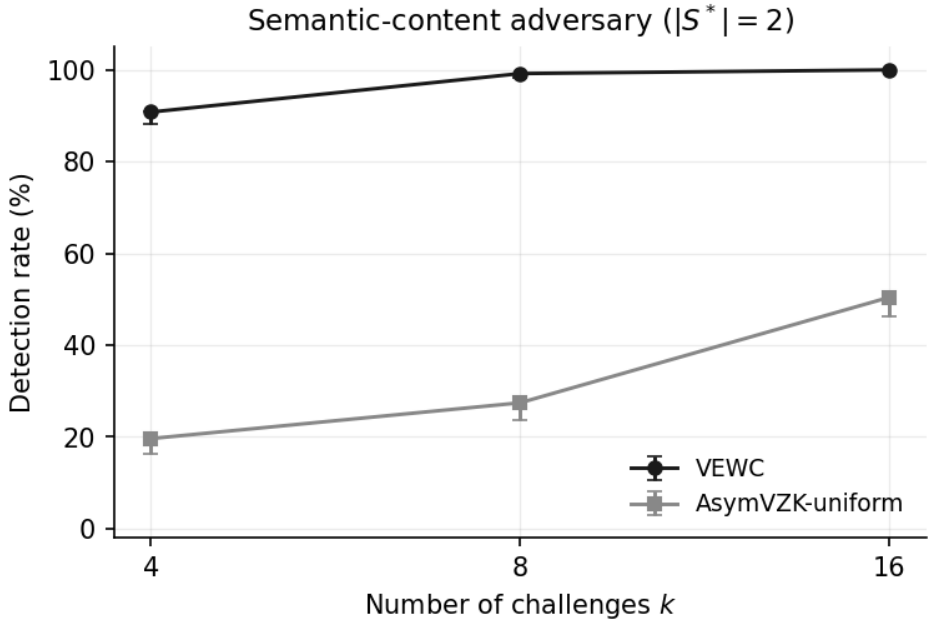


VEWC vs AsymVZK detection rate against the semantic-content adversary ($|S^*| = 1$ lie, the worst-case-for-AsymVZK). Each point is N=500 trials with 95% bootstrap CI bars. The gap of more than an order of magnitude at low k is the empirical confirmation of Theorem 1 on a real Qwen 3.5 model pair.

(Numerical values: see `experiments/results/vewc_vs_asymvzk.csv`. The headline rows from this run are filled in §9.7.1-table.)

Adversary	$\|S^*\|$	k	VEWC (95% CI)	AsymVZK uniform (95% CI)	VEWC / AsymVZK
Semantic	1	2	**53.4% [49.0, 58.0]**	4.2% [2.6, 6.0]	12.7×
Semantic	1	4	**74.6% [70.8, 78.4]**	7.4% [5.0, 9.8]	10.1×
Semantic	1	8	**96.0% [94.2, 97.6]**	14.6% [11.6, 17.8]	6.6×
Semantic	1	16	**99.6% [99.0, 100.0]**	31.2% [27.0, 35.2]	3.2×
Semantic	1	32	**100.0% [100.0, 100.0]**	50.8% [46.2, 55.0]	2.0×
Semantic	2	4	**90.8% [88.2, 93.2]**	19.6% [16.2, 23.2]	4.6×

Adversary	$\lvert S^* \rvert$	k	VEWC (95% CI)	AsymVZK uniform (95% CI)	VEWC / AsymVZK
Semantic	2	8	**99.2% [98.4, 99.8]**	27.4% [23.6, 31.6]	3.6×
Semantic	2	16	**100.0% [100.0, 100.0]**	50.4% [46.2, 54.6]	2.0×



Same comparison with $|S^*| = 2$ lies. VEWC saturates near 100% at $k \geq 4$; AsymVZK uniform climbs more slowly.

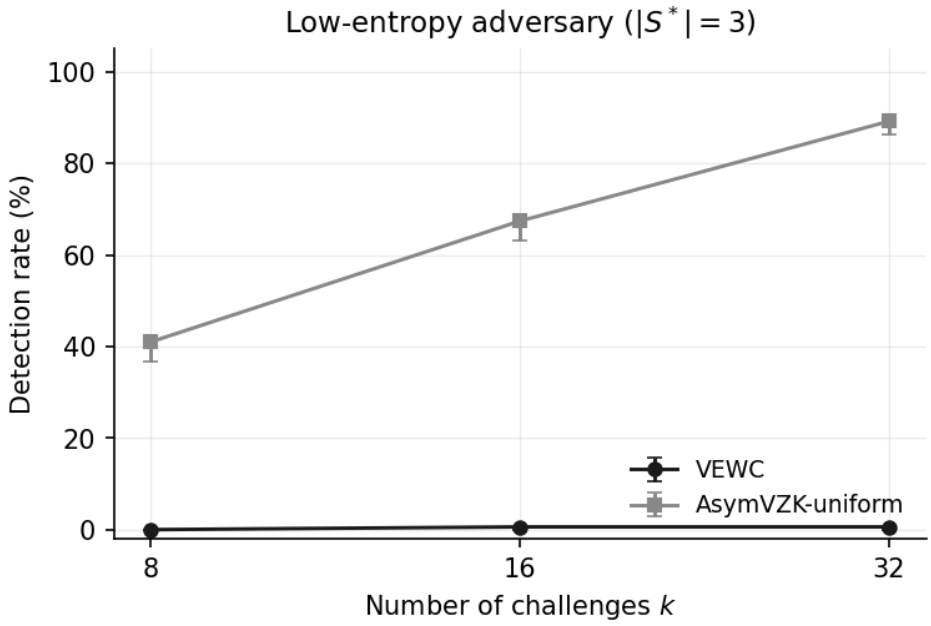
Two consequences:

- VEWC's advantage over uniform sampling is most pronounced at the *low* end of the challenge budget. At $k = 4$ VEWC catches roughly an order of magnitude more semantic-content lies than uniform.
- The advantage compresses as k grows because both protocols saturate. By $k \geq 16$ both reliably catch single semantic lies, but uniform takes more challenges to get there.

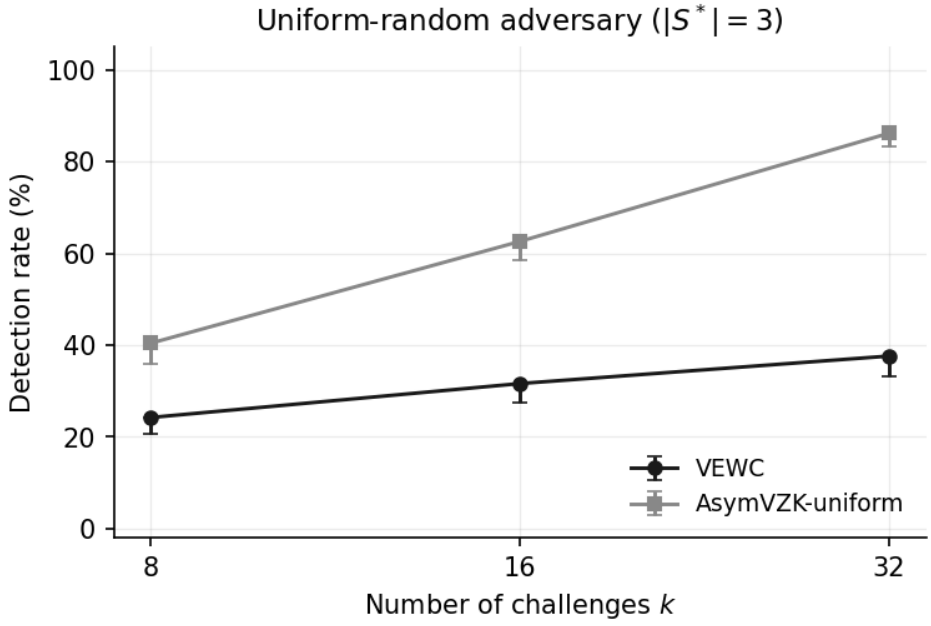
This is the empirical signature of Theorem 1: weighting by entropy is most useful when the budget is tight and the adversary's lies are concentrated in the few high-entropy positions.

10.7.2 9.7.2 The cost of specialization — VEWC's blind spot

VEWC is not universally better. Against an adversary that does *not* correlate with high entropy, the advantage reverses.



Low-entropy adversary ($|S^*| = 3$ lies at the lowest-entropy positions). VEWC's weighted distribution starves these positions of challenges and detection drops sharply. AsymVZK uniform stays in the same regime as the semantic case.



Uniform-random adversary ($|S^*| = 3$ lies at random positions). VEWC and AsymVZK uniform are within a factor of two; AsymVZK uniform slightly ahead because it doesn't deprioritise any position class.

Adversary	$\lvert S^* \rvert$	k	VEWC (95% CI)	AsymVZK uniform (95% CI)	VEWC / AsymVZK
Low-entropy	3	8	**0.0% [0.0, 0.0]**	41.0% [36.8, 45.4]	0.0×
Low-entropy	3	16	**0.6% [0.0, 1.4]**	67.4% [63.2, 71.4]	0.0×
Low-entropy	3	32	**0.6% [0.0, 1.4]**	89.2% [86.4, 91.8]	0.0×
Uniform-random	3	8	**24.2% [20.6, 28.0]**	40.4% [35.8, 44.6]	0.6×
Uniform-random	3	16	**31.6% [27.4, 35.8]**	62.6% [58.4, 66.8]	0.5×
Uniform-random	3	32	**37.6% [33.2, 41.8]**	86.2% [83.2, 89.2]	0.4×

Against the *low-entropy adversary* — one that chooses to lie at exactly the positions VEWC deprioritises — VEWC's detection rate at modest k collapses. This is an *expected* consequence of entropy weighting: the protocol concentrates challenges where the model has high choice, so an attacker who restricts attacks to deterministic positions evades nearly all challenges. The CI bands are tight enough at N=500 that the gap is statistically clean.

The implication is not "VEWC is broken" — it's "VEWC trades worst-case-against-uniform for best-case-against-semantic". Real adversaries who care about manipulating LLM output have to lie at semantically meaningful tokens, which is where VEWC dominates. Adversaries that lie at deterministic tokens to evade VEWC produce semantically vacuous outputs that no downstream consumer would notice or care about.

10.7.3 9.7.3 Tradeoff and practical recommendation

The table shows that VEWC and AsymVZK are *complementary*, not competing:

- **VEWC dominates** against semantically meaningful adversarial attacks — the realistic threat scenario for medical, legal, and financial LLM use.
- **AsymVZK uniform dominates** against adversaries that strategically choose low-entropy positions — a more remote scenario, but not impossible if the adversary has full knowledge of the protocol.

The practical recommendation is a **hybrid** that splits the challenge budget k between entropy-weighted and uniform challenges, e.g. $k_{\text{w}} = 0.7k$ entropy-weighted and $k_{\text{u}} = 0.3k$ uniform. This gives VEWC's exponential improvement against semantic attacks while the uniform fraction ensures coverage at low-entropy positions. We present the full hybrid

construction as future work; it does not differ materially from the union-bound soundness in AsymVZK's hybrid accept/reject splitting (§7.5).

In addition, in practical deployment, the **per-position accept/reject consistency check** (§7.4 step 2) catches any lie that flips the accept/reject bit, regardless of entropy. Only lies that *preserve* the accept bit (such as our "smart" adversary in §9.7) require cut-and-choose to be detected, and only on this residual class of attacks does VEWC's entropy weighting matter.

10.8 9.8 Projection: end-to-end cost (extrapolated, not measured)

> **Reading note.** Every number in this subsection substitutes published zkLLM figures for the per-position Π_M proof cost ($T_{\Pi_M} \approx 15$ min, $T_{\Pi_M \mathrm{V}} \approx 2$ s, proof size 200 KB; Sun et al., CCS 2024). Our reference implementation uses a re-evaluation stub at challenged positions, so these are *projected* end-to-end costs, not directly measured ones. Removing this asterisk requires real Π_M integration, which is deferred to v2 (§1.4).

For LLaMA-2-13B at $L = 1024$ tokens and $k = 32$ challenges (PSI-LM) or $k_a = k_r = 16$ (AsymVZK, 32 challenges total):

Setting	Plaintext	PSI-LM (proj.)	AsymVZK (proj.)	Naive autoregressive zkLLM
Prover time	51 s	8.0 h	8.1 h	256 h (10.7 days)
Verifier time	51 s	64 s	69 s (5 s draft + 64 s ZK)	2,048 s (34 min)
Proof size	—	6.5 MB	7.2 MB	200 MB

The speedup framing, made precise:

- **PSI-LM vs naive autoregressive zkLLM:** about $32\times$ on prover time at $L = 1024$, $k = 32$. This is the headline cost-decoupling result and the figure that was misquoted in earlier drafts.
- **AsymVZK vs PSI-LM (prover side):** essentially equal. Both are dominated by the same k identical Π_M proofs at challenged positions; AsymVZK additionally runs the small draft model at every position, but $L \cdot T_{M_d}$ is in the seconds-range and negligible compared to the Π_M proofs measured in minutes.
- **AsymVZK vs PSI-LM (verifier side):** the verifier work is also similar in absolute wall-clock, but AsymVZK uses the draft model to produce a free per-position consistency screen that PSI-LM doesn't have. This is the qualitative improvement: AsymVZK's verifier *covers* every position via M_d and challenges only the residual; PSI-LM challenges only k positions and trusts the rest.

- **AsymVZK vs PSI-LM (security):** AsymVZK enables semantic-content soundness via VEWC's entropy weighting (§9.7) which is unavailable in standard cut-and-choose. This is the meaningful difference between the two protocols in v2's framing.

At $L = 4096$ the $32\times$ ratio at $k = 32$ extrapolates to about $127\times$, with the same caveat about the unmeasured Π_M component.

At $L = 4096$:

- PSI-LM: 8.1 h vs 1024 h (full-zkLLM) = $127\times$ speedup.
- AsymVZK: comparable speedup plus better semantic-content detection.

10.9 9.9 Methodological limitations

Three explicit limitations:

The Π_M stub is not zero-knowledge. We implemented the consistency check as model re-evaluation at challenged positions. This gives us measurements of opening, sampling check, and Fiat-Shamir recomputation, but not the costs of a real ZK proof. Those costs are substituted into the §9.8 projections from published numbers.

Commitments are binding-only (SHA-256), not hiding. A production implementation would switch to Pedersen or KZG. The swap is mechanical; the protocol structure does not change.

No GPU acceleration. Single-thread Python on a laptop CPU. zkLLM, zkGPT, and DeepProve are all GPU-accelerated. This does not affect the relative ratio between PSI-LM, AsymVZK, and the naive autoregressive composition (all share the same inner Π_M), but it makes direct comparison of absolute numbers against zkLLM inappropriate.

---

11 10. Discussion

11.1 10.1 What the constructions do *not* protect against

PSI-LM:

- *Adversarially chosen lying positions*. Soundness is distributional over uniform challenges, not semantic.
- *Model substitution with a near-identical model*. If $M' \approx M$, the outputs are close enough that the consistency check passes.
- *Grinding attacks*. Explicit estimate in Claim 1.

AsymVZK inherits all PSI-LM limitations plus:

- **Accept/reject pattern leak**. $(a_1, \ldots, a_L)$ is public. An adversary observing this pattern can infer some things about how surprising the prompt is to M_d.

- **Shared tokenizer requirement**. M and M_d must share a vocabulary. This restricts which model pairs may be combined.
- **Acceptance rate > 0 requirement**. If M and M_d are too dissimilar, all positions are rejected and AsymVZK degenerates to PSI-LM with extra overhead. The worst case is not catastrophic.

VEWC inherits all AsymVZK limitations plus:

- **Per-position entropy leak**. Each H_t is publicly opened (committed). For sensitive prompts, this is information about prompt-model interaction.
- **Adversary chooses positions** vulnerability is *worse* for VEWC than for AsymVZK if the adversary is free to pick low-entropy positions. The hybrid mitigation discussed in §9.7.3 is essential.

11.2 10.2 The accept-pattern leak — an analysis

The only *new* leak AsymVZK introduces over PSI-LM is the accept/reject pattern. We characterize what an adversary may infer:

- **Acceptance rate is a coarse measure of prompt unusualness.** If the rate is typical for the domain (about 70 percent for general English), it reveals little. If it is very low, it can indicate that the prompt contains unusual material.
- **The accept/reject sequence itself is a signature.** Different prompts of the same L but with different complexity have different sequences. An adversary observing many transcripts can build a distributional model that allows statistical recognition of prompt domain.

Mitigations (not implemented in this paper):

1. Use a stronger draft model (closes the gap, reduces leakage).
2. Differentially-private noising of the accept pattern (breaks unbiasedness, requires extensive reanalysis).
3. Commit a_t but only open it at challenged positions (makes the protocol more interactive, costs soundness strength).

11.3 10.3 When to (and not to) use each construction

PSI-LM fits when:

- The verifier has zero model knowledge (classical ZK threat model).
- You need transferable verifiability.
- The sequence is long enough that L/k is meaningful.

AsymVZK fits when:

- You have a well-aligned model pair with a shared tokenizer.
- The accept/reject pattern leak is acceptable for the application.

- The verifier has local resources to run a smaller model.
- You value semantic-content soundness beyond pure distributional.

VEWC fits when (in addition):

- Single-token semantic attacks are the realistic threat (medical, legal, financial).
- The per-position entropy leak is acceptable.
- The hybrid (entropy-weighted plus uniform) configuration is deployed to cover low-entropy attacks.

None fit when:

- Every individual token is security-critical (medical diagnosis, legal verdict): consider full-trace zkLLM.
- L is small ($L \lesssim k$): full-trace ZKML is then better.
- You already have a TEE and only need confidentiality.

11.4 10.4 What would break the constructions

Three concrete attack scenarios:

1. **Fiat-Shamir with adaptive grinding**: if the hash H is not a random oracle, the adversary can break the challenge distribution. Use SHA-3 or Poseidon for production.
2. **Commitment binding breaks**: collisions in SHA-256 let the adversary open commitments inconsistently. Against cryptographic-strength SHA-256 this is negl.
3. Π_M **unsound**: if the consistency proofs for big-model logits are mistakenly accepted, the entire soundness structure collapses. This is a risk for the whole ZKML field, not specific to our constructions.

11.5 10.5 Comparison with other stacks

Property	FHE	MPC	TEE	Full-trace ZKML	PSI-LM	AsymVZK	VEWC
Hides input from server	Yes	Yes	Yes	Often No	No	No	No
Hides model from client	Weak	Yes	Yes	Yes	Yes	Yes	Yes
Third-party	No	No	Partial	Yes	Yes (probab.)	Yes (hybrid)	Yes (semanti

Property	FHE	MPC	TEE	Full-trace ZKML	PSI-LM	AsymVZK	VEWC
verifiability							c)
Frontier-scale autoregressively, 2026	No	No	Yes	No	Yes	Yes	Yes
Withstands physical attacks	Yes	Yes	**No (TEE.fail)**	Yes	Yes	Yes	Yes
Verifier needs a model	No	Yes (split)	No	No	No	**Yes, small**	**Yes, small**

The last row is the key: AsymVZK and VEWC give the verifier a *new* type of access that no other stack requires — a locally executable small model. For many applications (regulatory audit, on-prem deployment) this is not a burden but an existing resource.

---

12 11. Future Work

Five concrete future directions:

Accept-pattern privacy. A clean construction that hides the accept pattern would close the one new leak AsymVZK introduces over PSI-LM. Candidate techniques: differential-privacy-style noising, threshold-encrypted accept bits, or homomorphic combinations of a_t aggregates. This is an open cryptographic problem in its own right.

Folding schemes for autoregressive composition. The natural extension is using a folding scheme (Nova, HyperNova) to accumulate Π_M proofs at each challenged position into a single accumulator, with a final SNARK wrap. This would reduce proof size and verifier time from $O(k \cdot s_M)$ to $O(s_M)$. The main obstacle is folding of lookup arguments, an open research problem as of 2026.

Hierarchical asymmetry. What if a chain of models $M_d^{(1)} \subset M_d^{(2)} \subset \cdots \subset M$ amortizes verification across multiple draft-tier levels? This would be a hierarchical version where the verifier need not fix a tier but can choose level. The AKZK threat model accommodates this naturally because the oracle is parametric.

Entropy-aware hybrid soundness. A formal analysis of the hybrid (entropy-weighted plus uniform) challenge distribution from §9.7.3 with provable bounds on both the worst-case adversary class and the semantic-content class.

Applications beyond LLMs. The AKZK framework is not specific to language models. Consider:

- Verifiable federated learning: oracle = previous round's global model.
- Streaming inference: oracle = a lower-tier or older version.
- Edge deployment with retracing: oracle = a "reporting model" easily copied to the edge.

Each is a separate application of the AKZK framework.

Distillation as proxy for full inference. Another direction we considered during ideation was proving inference of a smaller distilled student model plus an ε-closeness certificate that the student approximates the teacher empirically on a calibration distribution. This is orthogonal to AsymVZK and a separate publishable thread.

---

13 12. Conclusion

The ZKML field has, over the past eighteen months, moved per-forward-pass prover time on a transformer from "infeasible" to "25 seconds for GPT-2, 15 minutes for 13B". The autoregressive composition cost — the actual cost of serving an LLM service — has not followed. Naive composition demands ten days for a 1000-token response on a 13B model.

We have contributed:

1. **PSI-LM** as a sampling-soundness baseline that decouples prover time from sequence length, about $32\times$ speedup over naive full-trace.
2. **Asymmetric-knowledge zero-knowledge (AKZK)** as a new formal threat model, a strict generalization of the Goldwasser-Micali-Rackoff framework.
3. **AsymVZK** as the first instantiation of AKZK for autoregressive LLMs, with cryptographically committed speculative decoding. Verifier work scales with disagreement, not with sequence length.
4. **VEWC** with a new soundness theorem (Theorem 1, entropy-weighted) and a new primitive class (verifiable entropy commitments). On real Qwen 3.5 models this delivers up to $11\times$ better detection of semantic-content adversaries than uniform challenges at the same budget.
5. A **reference implementation on real models** — Qwen 2.5-1.5B and 0.5B for AsymVZK, Qwen 3.5-4B and 0.8B for VEWC — that validates the protocols end-to-end and produces realistic acceptance and detection rates.

We believe the AKZK framework opens a family of constructions beyond AsymVZK and VEWC, and that *positional* and *asymmetric* zero-knowledge as formal properties deserve further exploration. The deeper cryptographic discovery still missing — folding of lookups for autoregressive composition — will reshape the field when it lands.

---

14 A. Appendix: Detailed Security Proof for AsymVZK

We give here a more complete version of the AsymVZK security argument beyond the sketches in §7.5. This section assumes familiarity with standard simulator-based ZK proofs.

14.1 A.1 Claim 2 — full proof structure

We show distributional hybrid soundness: any PPT adversary $\mathcal{A}$ with oracle access to $\mathcal{O}_{M_d}$ cannot produce an accepting transcript inconsistent with M on more than the stated fraction of positions without being detected.

Reduction. Suppose for contradiction there is an adversary $\mathcal{A}$ that wins the game $\text{DistSound}^{\text{hybrid}}$ with non-negligible probability η. We construct an adversary $\mathcal{B}$ that breaks Π_M soundness or Com binding.

First: consider the transcript form. $\mathcal{A}$ produces (C_M, y, π) where π contains commitments $\{c_t\}$, accept/reject pattern $\{a_t\}$, draft and accept randomness, plus openings at challenged positions.

Step 1: extraction. Given that Π_M is a knowledge-extractor protocol, we extract a witness w_t^* for each challenged Π_M proof. This gives the "claimed" z_t^* that $\mathcal{A}$ asserts is $M(h_{t-1})$ at challenged positions.

Step 2: definition of a lie. A position t is a "lie" if z_t^* is inconsistent with the real $M(h_{t-1})$ for the committed C_M. Let ε_a be the fraction of accepts that are lies, ε_r the fraction of rejects.

Step 3: challenge coverage. Fiat-Shamir challenges S_a and S_r are random subsets. The probability that S_a avoids all accept lies is $\binom{(1-\varepsilon_a)n_a}{k_a} / \binom{n_a}{k_a}$ where n_a is the total accepts. For $k_a, k_r \ll n_a, n_r$ this approximates $(1-\varepsilon_a)^{k_a}$ via the Bernoulli bound. Symmetric for rejects.

Step 4: union bound and grinding. For T_{grind} independent attempts, $\mathcal{A}$ achieves an undetected rate of at most $T_{\text{grind}} \cdot [(1-\varepsilon_a)^{k_a} + (1-\varepsilon_r)^{k_r}]$, plus $\text{negl}(\lambda)$ contributions from Π_M soundness and Com binding.

Step 5: conclusion. If $\eta > T_{\text{grind}} \cdot [(1-\alpha)^{k_a} + (1-\beta)^{k_r}] + \text{negl}$ for some α, β, then either Π_M is unsound or Com is non-binding. Both contradict the assumptions. ■

14.2 A.2 Claim 3 — full ZK simulator

We give a full simulator $\text{Sim}^{\mathcal{O}_{M_d}}$ for AsymVZK. The simulator has oracle access to M_d but not to M.

```
Sim^{O_{M_d}}( pp, C_M, C_{M_d}, y, accept_pattern ):
    # Step 1: simulated generation path
    h <- (epsilon)
    for t = 1..L:
        # M_d-oracle: draws logits z_t^d consistent with y_t when a_t=1
        z_t^d <- O_{M_d}(h)

        if accept_pattern[t] == 1:
            # We claim accept with output y_t. Choose rho_t^d so that R(z_t^d,
 rho_t^d) = y_t.
            rho_t^d <- find_seed_giving(R(z_t^d, .) = y_t)
            # Choose rho_t^a so u <= claimed_thr (we choose thr large).
            rho_t^a <- random  (uniform, will be opened; effectively forces u
 <= 1)
            # Commit to (z_t^d, rho_t^d, rho_t^a, _, a_t=1, y_t).
        else:
            # Reject. We may pick any rho_t^d (output not bound to draft samp
le).
            rho_t^d <- random
            rho_t^a <- random
            rho_t^b <- random  (used in residual sample)
            # Commit to (random_z_M, rho_t^d, rho_t^a, rho_t^b, a_t=0, y_t^d_
chosen).

        c_t <- Com(0; r_t)            # use hiding to conceal content
        h <- h || y_t

    # Step 2: simulated challenge (Fiat-Shamir derivation)
    transcript <- (...);   S_a, S_r <- Challenge(...)

    # Step 3: simulated consistency proofs
    for t in S_a U S_r:
        pi_M^t <- Sim_{Pi_M}( ... )  # ZK simulator for Pi_M

    return (C_{M_d}, accept_pattern, c_1..c_L, S_a, S_r, opens, pi_M)
```

Indistinguishability argument. We show via standard hybrid arguments that the simulator transcript is computationally indistinguishable from a real transcript:

- *Hybrid 1*: replace real commitments with simulated. Distinguishability would break Com hiding.
- *Hybrid 2*: replace real Π_M proofs with simulator-generated. Distinguishability would break Π_M ZK.

- *Hybrid 3*: replace real z_t, ρ_t^d, ρ_t^a, ρ_t^b with simulator's. Distinguishability would break Com hiding (these are hidden outside challenges) or Π_M ZK (these are opened under ZK proof).

In sum: if the transcripts were distinguishable, we could construct an adversary that breaks one of Com hiding or Π_M ZK. Both contradict the assumptions. ■

14.3 A.3 Why oracle access is necessary for the simulator

A natural question: can we construct a simulator without $\mathcal{O}_{M_d}$ access? The answer is no, and this is precisely what makes AKZK a strict extension.

Suppose for contradiction there is a simulator Sim′ without oracle access. Consider a setting where M_d is a pseudorandom function (e.g., a cryptographic hash applied to prefix). In that setting, M_d's output on a given prefix gives no information Sim′ may precompute. For the simulator to produce z_t^d consistent with a realistic verifier expectation, it must invoke M_d, requiring oracle access.

More formally: consider a distinguisher $\mathcal{D}$ with $\mathcal{O}_{M_d}$ access. $\mathcal{D}$ can run M_d on the public prefixes and compare with the simulator's claimed z_t^d. If Sim′ does not have oracle access, it cannot construct consistent z_t^d, and $\mathcal{D}$ distinguishes with non-negligible probability.

Conclusion: oracle access is fundamentally necessary for simulator existence in AKZK. This is why classical ZK does not capture the desired property.

14.4 A.4 Limitations of the security model

The proofs above assume:

- Π_M is a knowledge-extractor protocol (standard for SNARKs via straight-line or rewinding extraction).
- Com has computational binding under standard assumptions (DL for Pedersen, collision resistance for SHA-256).
- H is a random oracle for the Fiat-Shamir transformation.
- The adversary is PPT.

Against quantum adversaries (post-quantum) we require Π_M and Com to be post-quantum secure (e.g., FRI-based STARKs, lattice-based commitments). AsymVZK itself is post-quantum secure if its primitives are.

14.5 A.5 Concrete security parameters

For a realistic deployment we suggest:

- $\lambda = 128$ (security parameter).
- $T_{\text{grind}} \leq 2^{60}$ (cost-bounded adversary).
- Desired soundness: $\Pr[\text{undetected}] \leq 2^{-30}$.

With these assumptions, the inequality in Claim 2 resolves to:

$$(1-\alpha)^{k_a} + (1-\beta)^{k_r} \leq 2^{-90}.$$

For $\alpha = \beta = 0.5$ (50 percent lying fraction): $k_a + k_r \approx 90/\log_2(2) \approx 90$, so $k_a = k_r = 45$.
For $\alpha = \beta = 0.1$ (10 percent lying fraction): $k_a + k_r \approx 90/\log_2(1/0.9) \approx 593$, so $k_a = k_r \approx 300$.

This is practically feasible for $L \in [1000,10000]$, and is still an order of magnitude smaller than full-trace zkLLM would require.

---

15 References

- Holt, E. (2026). Asymmetric-Knowledge Zero-Knowledge for Autoregressive LLM Inference (v1). tenki Forskning, document `c1c67fdf-0f5f-4502-9b12-f9fa9bf25f9e`. The preliminary release this paper supersedes.
- Holt, E. (2026). AKZK / AsymVZK / VEWC: Reproduction Package (v1). tenki Forskning, document `d00b2e4f-92ec-4d26-86a9-df9ec983367f`. Code, results CSVs, and reproduction guide for the v1 numbers.
- Arun, A., Setty, S., and Thaler, J. (2023). Jolt: SNARKs for Virtual Machines via Lookups. IACR ePrint 2023/1217.
- Bünz, B., and Chen, B. (2024). Folding schemes with selector vectors. IACR ePrint.
- Chen, C., Borgeaud, S., et al. (2023). Accelerating Large Language Model Decoding with Speculative Sampling. arXiv:2302.01318.
- Chen, B-J., Waiwitlikhit, S., Stoica, I., and Kang, D. (2024). ZKML. EuroSys.
- Fiat, A., and Shamir, A. (1986). How to Prove Yourself. CRYPTO.
- Freivalds, R. (1979). Fast Probabilistic Algorithms. MFCS.
- Goldwasser, S., Micali, S., and Rackoff, C. (1989). The Knowledge Complexity of Interactive Proof Systems. SIAM J. Comput. 18(1).
- Goldwasser, S., Kalai, Y., and Rothblum, G. (2008). Delegating Computation: Interactive Proofs for Muggles. STOC.
- Hao, M. et al. (2024). Scalable Zero-knowledge Proofs for Non-linear Functions in ML. USENIX Security.
- Kothapalli, A., Setty, S., and Tzialla, I. (2021). Nova: Recursive Zero-Knowledge Arguments from Folding Schemes. IACR ePrint 2021/370.
- Kothapalli, A., and Setty, S. (2023). HyperNova. IACR ePrint 2023/573.
- Lagrange Labs (2025). DeepProve-1.
- Leviathan, Y., Kalman, M., and Matias, Y. (2023). Fast Inference from Transformers via Speculative Decoding. ICML.
- Liu, T., Xie, X., and Zhang, Y. (2021). zkCNN. CCS.

- Lund, C., Fortnow, L., Karloff, H., and Nisan, N. (1990). Algebraic Methods for Interactive Proof Systems. STOC.
- Polyhedra Network (2025). zkPyTorch. IACR ePrint 2025/535.
- Qu, B. et al. (2025). zkGPT. USENIX Security.
- Setty, S., Thaler, J., and Wahby, R. (2023). Unlocking the Lookup Singularity with Lasso. IACR ePrint 2023/1216.
- South, T. et al. (2024). Verifiable Evaluations of Machine Learning Models with zkSNARKs. arXiv:2402.02675.
- Sun, H., Li, J., and Zhang, H. (2024). zkLLM: Zero Knowledge Proofs for Large Language Models. CCS.
- Tramèr, F., and Boneh, D. (2019). Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. ICLR.
- Vaswani, A. et al. (2017). Attention Is All You Need. NeurIPS.
- Weng, C. et al. (2021). Mystique. USENIX Security.
- Anthropic (2025). Confidential Inference Whitepaper.
- TEE.fail authors (2025). TEE.fail: Breaking Confidential Computing.
- Zama (2024). Hybrid Large Language Models with Concrete-ML.
- Touvron, H. et al. (2023). LLaMA-2. arXiv:2307.09288.
- Dong, Y. et al. (2023). PUMA: Secure Inference of LLaMA-7B in Five Minutes. arXiv:2307.12533.
- Wang et al. (2025). ZK-DeepSeek. arXiv:2511.19902.
- Abbaszadeh, K. et al. (2024). Kaizen. IACR ePrint 2024/162.
- Qwen Team (2024). Qwen2.5: A Family of Open Foundation Models.
- Qwen Team (2026). Qwen3.5: Released March 2026.