INCREMENTAL VERIFIABLE INFERENCE FOR STATE-SPACE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) are widely deployed as remote services, where users observe only the inputs and outputs and cannot directly check whether the advertised model was executed faithfully. This challenge becomes more pronounced for long autoregressive generations, where an execution may span thousands of dependent tokens and proving the full sequence can become increasingly expensive. We present MPROOF, an incremental proof system for end-to-end autoregressive inference in state-space language models at the hundred-million-parameter scale. Our design builds on two properties of modern state-space models. Their recurrent state has a fixed size independent of sequence length, providing a compact interface between successive chunks; state-space duality (SSD) additionally represents the computation within a chunk as parallel matrix operations. MPROOF exploits these two structures to verify completed chunks jointly and extend an existing proof as generation proceeds, without revisiting earlier model computation.

On a 187M-parameter Mamba-3 model, MPROOF proves 4,096 tokens in 22.85 minutes using 128-token chunks, with prepared verification taking 0.451 seconds. Across a $16\times$ increase in sequence length, total prover time grows by only $13.6\times$, while prepared verification stays below 0.5 seconds and cryptographic proof size is unchanged. Our code is available at https://anonymous.4open.science/r/MProof-6D18/.

1 INTRODUCTION

When a language-model service appears to become less capable, how can its users tell what happened? A study of GPT-3.5 and GPT-4 documented task-dependent behavioral drift across service versions (Chen et al., 2023); Anthropic later traced degraded Claude responses to infrastructure bugs that its evaluations had missed (Anthropic, 2025b). Emerging deployments make the execution path even harder to oversee: IEEE 802.11's AI Offload effort, for example, is exploring inference on Wi-Fi access points and other edge devices (IEEE 802.11 Working Group, 2026). Behavioral evaluations can reveal quality changes, and model fingerprints can help identify the model behind a service (Xu et al., 2024), but neither establishes that a particular response follows the requested computation. Hardware attestation can provide integrity, but requires trusted hardware (Tramèr & Boneh, 2019); Section 2.1 compares these auditing approaches with cryptographic proofs. The resulting question is not only whether a model performs well, but whether the provider faithfully applies that model and its agreed inference procedure to each request.

Zero-knowledge proofs offer a cryptographic solution to this execution-consistency problem. The provider proves that the output follows the agreed model, numerical rules, and prompt; the client checks the proof without rerunning inference. Zero knowledge can also protect model weights. Recent research has made such proofs feasible for language models (Sun et al., 2024; Qu et al., 2025; Wang et al., 2026). Long agent interactions accumulate instructions, reasoning, and tool results (Anthropic, 2025a; Hadeliya et al., 2025), requiring this guarantee throughout generation. Proving tokens separately repeats substantial work. zkAgent instead uses causal masking to check an entire completed transcript in one forward pass, organizing the computation across tokens as in prefill (Wang et al., 2026). Token activations then grow with transcript length T, and materialized full-attention tensors contain $\Theta(T^2)$ entries. Unpredictable generation lengths thus make prover memory demand vary substantially across requests.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Incrementally verifiable computation (IVC) instead extends a proved prefix with new computation, without reproving the prefix (Kothapalli et al., 2022; Kothapalli & Setty, 2024). Its effectiveness depends on the update boundary: a full-attention Transformer still maintains a growing KV cache that every new query accesses. Recursion alone does not bound this model-state interface. Recurrent networks supply a fixed-size interface; zkRNN recursively proves their individual steps (Zarinjouei et al., 2026). Modern state-space models make this structure relevant to language modeling: Mamba-3 achieves competitive quality against parameter-matched Transformers while improving state tracking (Lahoti et al., 2026). It combines a fixed-size recurrent state for decoding with state-space duality (SSD), which exposes matrix operations for efficient parallel prefill (Dao & Gu, 2024; Lahoti et al., 2026). We observe that fixed state bounds what each proof update inherits, while SSD enables sharing within the new computation. The remaining choice is granularity: single-token updates repeat recursion overhead, while sequence-wide proofs tie the working set to the full history. How much computation should each recursive proof cover?

Our approach. We present MPROOF, the first incremental proof system for end-to-end autoregressive inference in state-space language models at the hundred-million-parameter scale. Our *chunkwise proving* architecture extends long-running generation through bounded updates. Its *wide arithmetization* shares operator checks and SSD contractions, replacing repeated proof work and serial reduction stages with wide tensor relations. Optimized *reduction before recursive composition* keeps model activations outside recursion. *Sequence binding* connects this computation to the actual prompt and generated tokens. Together, these designs make proof work scale with newly processed tokens rather than the accumulated history.

Our main contributions are as follows.

- We enable proof work to be shared across generated tokens while keeping each update bounded as generation continues. Chunkwise proving groups completed tokens; wide arithmetization jointly checks their repeated operators and recurrent computations. We prove that these checks preserve the specified fixed-point execution.
- We provide end-to-end verification that the public output follows the committed model, prompt, and decoding rule. We adapt reduction before recursive composition to keep model activations outside recursion and strengthen sequence binding to authenticate the actual tokens. Prover cost grows linearly, while cryptographic proof size remains constant at fixed chunk width.
- We implement MPROOF on a pretrained 187M-parameter Mamba-3 model. Proving 4,096 tokens takes 22.85 minutes, with verification below 0.5 seconds across a $16\times$ length increase. Token sharing yields $1.79\times$ faster proving and 47.2% lower memory; SSD sharing accelerates core proof routines by $4.55\times$. Numerical and task evaluations show close agreement with BF16.

2 BACKGROUND

2.1 AUDITABLE MACHINE LEARNING

Auditing methods provide different kinds of evidence. Behavioral audits such as RUT use black-box queries to detect changes from a reference model (Zhu et al., 2026); instructional fingerprinting identifies model provenance from enrolled response signatures (Xu et al., 2024). Trusted execution binds outputs to an authenticated model through attestation (Tramèr & Boneh, 2019); cryptographic proofs bind them to committed parameters without trusted hardware. zkLLM and zkGPT target Transformer forward proofs (Sun et al., 2024; Qu et al., 2025), zkRNN recursively proves recurrent steps (Zarinjouei et al., 2026), and zkAgent jointly proves generation transcripts through causal masking (Wang et al., 2026). Table 1 summarizes their complementary capabilities.

Here, *API-only* denotes query-only access to the target; *drift* tests output-distribution changes. *Identity* covers behavioral identification (IF) or authenticated model binding (Slalom and proofs); *integrity* concerns each execution. For generation proofs, *E2E* spans input tokens to output selection, *joint* checks multiple generated positions together, and *Inc.* extends a proof state without redoing prior model computation. MPROOF combines all three.

Table 1: Capability matrix for representative approaches to auditable model services.

Approach	System	Auditing capability				Generation proof		
		API-only	Drift	Identity	Integrity	E2E	Joint	Inc.
Behavioral audit	RUT	✓	✓	×	×	–	–	–
Fingerprinting	IF	✓	×	✓	×	–	–	–
Trusted execution	Slalom	×	×	✓	✓	–	–	–
Cryptographic proofs	zkLLM	×	×	✓	✓	×	×	×
	zkGPT	×	×	✓	✓	×	×	×
	zkRNN	×	×	✓	✓	×	×	✓
	zkAgent	×	×	✓	✓	✓	✓	×
	MPROOF	×	×	✓	✓	✓	✓	✓

✓ supported × not provided – outside scope

2.2 AUTOREGRESSIVE INFERENCE OF STATE-SPACE MODELS

An autoregressive language model predicts each output token from the prompt and previous outputs (Radford et al., 2019). Given prompt $\mathbf{u}$, it assigns output $\mathbf{y} = (y_1, \ldots, y_M)$ probability $p_\theta(\mathbf{y} \mid \mathbf{u}) = \prod_{i=1}^{M} p_\theta(y_i \mid \mathbf{u}, y_{<i})$. *Prefill* processes the prompt; *decoding* repeatedly selects the next token and updates the model state. Full-attention Transformers store earlier keys and values, whereas recurrent state-space models maintain a state of fixed size (Gu & Dao, 2024; Dao & Gu, 2024).

For one head of the single-input, single-output (SISO) Mamba-3 model, the exact-arithmetic recurrence is (Lahoti et al., 2026)

$$H_t = \alpha_t H_{t-1} + \beta_t(\widehat{B}_{t-1} \otimes X_{t-1}) + \gamma_t(\widehat{B}_t \otimes X_t), \qquad Y_t = \widehat{C}_t^{\mathsf{T}} H_t. \tag{1}$$

Here $H_t \in \mathbb{R}^{d_s \times d_v}$ is the state, $\widehat{B}_t, \widehat{C}_t \in \mathbb{R}^{d_s}$ absorb the input-dependent rotations, and $X_t, Y_t \in \mathbb{R}^{d_v}$; $\otimes$ denotes an outer product. The coefficients weight the retained state and previous/current inputs. SSD expresses the block response as

$$Y = Y^{\mathsf{carry}} + (L \odot \widehat{C}\widehat{B}^{\mathsf{T}})X, \tag{2}$$

where the causal matrix L weights source–target pairs, $\odot$ denotes elementwise multiplication, and the carry incorporates the incoming state and preceding input pair (Dao & Gu, 2024; Lahoti et al., 2026). SSD thus combines parallel block computation with a fixed recurrent boundary.

2.3 NUMERICAL PROGRAM AND PROOF PRIMITIVES

Arithmetic proof systems operate over finite fields, so we represent neural computations by bounded fixed-point integers with explicit rounding. MPROOF proves the resulting program, ChunkFP. Its specification spec fixes the model, function tables, chunk width, initial state, decoding rule, scales, and contraction order. Signed integers are embedded in $\mathbb{F} = \mathbb{F}_p$. A rescale computes $q = \mathsf{RS}_s(z) = \lfloor (z + 2^{s-1})/2^s \rfloor$ and checks $z + 2^{s-1} = 2^s q + r$, $0 \le r < 2^s$; operand ranges exclude modular wraparound. Changing chunk width may change the rounding schedule. Section 4.4 evaluates the resulting programs against BF16.

A tensor's *multilinear extension* (MLE) matches its entries at Boolean indices. Sumcheck reduces tensor sums to evaluations; GKR applies these reductions through arithmetic layers, and LogUp checks function and range lookups (Lund et al., 1992; Goldwasser et al., 2008; Haböck, 2022). Polynomial commitments bind the tensors. Accumulation combines evaluation claims for final opening; folding combines recursive verifier instances for a final validity check, or *decider* (Kothapalli et al., 2022; Kothapalli & Setty, 2024). Appendix B collects notation and protocol details.

2.4 SECURITY MODEL

A client $\mathcal{V}$ outsources inference to an untrusted provider $\mathcal{P}$. Statement x fixes model/table commitments, spec, public tokens, execution label, and setup identity. The client authenticates the commitments and setup; the provider controls intermediate values and proof messages. Let $\mathcal{R}_{\mathsf{traj}}(x; w) = 1$ denote a valid ChunkFP execution from the prescribed initial state to the public output. We require:

- **Completeness:** honest executions within the registered ranges are accepted.
- **Execution integrity:** for any efficient malicious provider $\mathcal{P}^*$, $\Pr[\mathsf{Verify}(x, \Pi) = 1 \wedge \nexists w : \mathcal{R}_{\mathsf{traj}}(x; w) = 1] \leq \epsilon$.

Theorem 5 composes the native checks with recursive verification in the random-oracle model. Appendix C.4 states its component assumptions and cites the corresponding composition and Fiat–Shamir results. The measured protocol is non-hiding; Appendix C.6 gives an optional zero-knowledge construction.

3 MPROOF

MPROOF introduces chunkwise proving (Section 3.1) and wide arithmetization (Section 3.2), then optimizes reduction before recursive composition (Section 3.3). Section 3.4 integrates sequence binding into the end-to-end protocol; Section 3.5 analyzes its complexity.

3.1 CHUNKWISE PROVING

The proof unit controls shared work and recursive overhead. Independent token proofs repeat operator checks; sequence-wide proofs grow with the history. We jointly prove each completed C-token *chunk* and extend the preceding proof with one recursive update. For T processed tokens this gives $K = T/C$ updates, with $C = 2^c$. Chunk k checks $(\mathbf{s}_{k+1}, \mathbf{o}_k) = \mathsf{ChunkFP}_{\kappa_k}(\mathbf{s}_k, \mathbf{t}_k; \theta, \mathcal{T})$, where κ_k selects prefill or decode, $\mathbf{t}_k, \mathbf{o}_k$ are its input/output tokens, and $\theta, \mathcal{T}$ fix the parameters and lookup tables shared by all chunks.

For N_L layers, $\mathbf{s}_k$ contains each layer's recurrent state $H_{k,\ell}$, phase $\Psi_{k,\ell}$, and previous input pair $(B^-_{k,\ell}, X^-_{k,\ell})$, with $\ell < N_L$. Each chunk reads only this fixed-size boundary and new tokens. A packed commitment $\mathsf{com}^{\mathsf{st}}_k$ identifies the same state on both sides of each chunk boundary. The public initial state fixes $\mathbf{s}_0$.

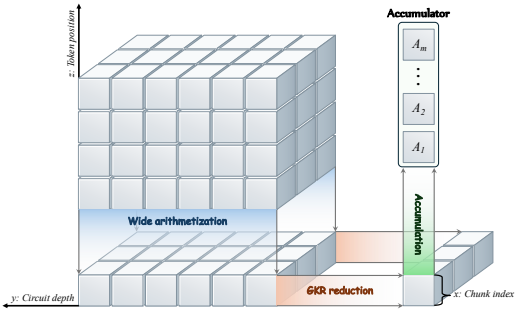

Figure 1: Joint proving within chunks; compact claims and recurrent boundaries across chunks.

Grouping precedes proof construction. Decode generates tokens causally, then its completed trajectory exposes all relations for joint checking (Figure 1). Prefill uses the same unit. Decode additionally proves token selection and output-to-input links. The final prompt token is the first decode input, so $T = |\mathsf{Prompt}| + |\mathsf{Output}| - 1$. The evaluated interface exposes verification checkpoints at full-chunk boundaries. Appendix B.3.1 defines a length-bound padding extension for shorter final outputs, retaining width C.

3.2 WIDE ARITHMETIZATION

Token-wide checks. Repeated operators become one tensor relation indexed by token and feature coordinates. Let $P_g(t, b)$ be an operator residual, with $b \in \{0,1\}^v$ and $t \in \{0,1\}^c$. For pointwise multiplication, $P_g = \widetilde{X}\widetilde{Y} - \widetilde{Z}$. Once the chunk tensors are fixed, a random point $\mathbf{r} = (r_t, r_b)$ defines

$$0 = \sum_{t \in \{0,1\}^c} \sum_{b \in \{0,1\}^v} \widetilde{\mathsf{eq}}(r_t, t)\widetilde{\mathsf{eq}}(r_b, b) P_g(t, b), \qquad Cv \longrightarrow v + \log_2 C \text{ rounds.} \tag{3}$$

Here $\widetilde{\mathsf{eq}}(a, b) = \prod_i (a_i b_i + (1-a_i)(1-b_i))$. One sumcheck now replaces C independent invocations. Projections similarly combine $y_t = W x_t$ into $Y = XW^{\mathsf{T}}$ and share the reduction to W. Lookups aggregate all queries $q_{t,i}$ against one table (τ_j) through

$$\sum_{t,i} \frac{1}{\beta + q_{t,i}} = \sum_j \frac{m_j}{\beta + \tau_j}, \qquad C(q + B) \longrightarrow Cq + B \text{ leaves,} \tag{4}$$

where q is the number of queries per token, B the table size, and m_j the aggregate multiplicity. A random challenge compresses function input/output tuples; GKR checks the fraction tree for each

lookup lane. The same tensor organization covers nonlinear functions and integer ranges. Sharing also removes the intermediate claims that separate token proofs must merge.

Chunk-wide recurrent checks. SSD expresses the in-chunk response through a Gram contraction, causal weighting, and a source contraction. For head j,

$$\begin{aligned} G^{(j)} &= \mathsf{RS}_{q_g}(\widehat{C}^{(j)}\widehat{B}^{(j)\mathsf{T}}), \quad M^{(j)} = \mathsf{RS}_{e_u}(\Omega^{(j)} \odot G^{(j)}), \\ Y^{\mathsf{intra},(j)} &= \mathsf{RS}_{q_y}(M^{(j)}X^{(j)}). \end{aligned} \tag{5}$$

The causal mask Ω supplies decay and input weights; q_g, e_u, q_y fix the rescaling schedule. Instead of reducing each target separately, we check the raw source contraction in one reduction:

$$\widetilde{Y}^{\mathsf{raw}}(r_t, r_j, r_e) = \sum_{s,j} \widetilde{\mathsf{eq}}(r_j, j)\widetilde{M}(r_t, s, j)\widetilde{X}(s, j, r_e). \tag{6}$$

Here r_t, r_j, r_e select target, head, and value dimensions; s, j range over Boolean source and head indices. One target evaluation combines all C outputs. The common head index stays inside the sum to preserve the product identity. The Gram contraction uses the same rule; carry readouts and state exports complete the transition.

The decay paths would otherwise add $C-1$ mask stages and C carry stages to the GKR reduction. We commit each rounded trajectory U with a shifted view U^-, constrained by $U_0^- = U^{\mathsf{in}}$ and $U_t^- = U_{t-1}$. Transitions such as $U_t = \mathsf{RS}_s(\alpha_t U_t^-)$ then form one wide relation. Mask and carry decays require two product/rescale stages in total, replacing a $\Theta(C)$-deep path. Seed, shift, and range checks enforce every original rounding step. Appendix B.4 gives the complete relations.

Lemma 1 (Execution-preserving wide arithmetization)**.** *For a fixed admitted ChunkFP specification with $C \geq 2$ and fixed model and table sources, the wide and positionwise relations admit the same model executions, output tokens, and outgoing states.*

Token widening reindexes the same residuals, so an assignment satisfies the tokenwise relations exactly when it satisfies their packed form. For each rounded recurrence, the boundary fixes the seed; rescale and range constraints determine each subsequent entry uniquely. Induction therefore reconstructs the same trajectory. Shared SSD contractions retain their operands and rescale sites. Lemma 2 bounds the randomized checks of these relations; Appendix C.1 gives the complete equivalence proof.

3.3 Reduction before Recursive Composition

We adapt GKR composition and evaluation accumulation (Abbaszadeh et al., 2024; Bünz & Chen, 2025) to recurse over the verifier rather than the model. A native chunk proof follows $(x_k, \pi_k) \xrightarrow{\mathsf{GKR/LogUp}} \mathcal{E}_k \xrightarrow{\mathsf{Accumulate}} \mathbf{a}_k$, where $\mathcal{E}_k = \{(f, \mathsf{com}_p, r, v) : \widetilde{p}(r) = v\}$ contains evaluation claims and $\mathbf{a}_k$ retains one claim per family. Family f fixes the tensor's role and domain. The recursive relation checks $\sigma_k = \mathsf{CheckUpdate}(\sigma_{k-1}, x_k, \pi_k, \mu_k, u_{k-1})$, with proof state σ_k, update messages μ_k, and pending instance u_{k-1}. Its witness contains proof messages and verifier intermediates, keeping model activations outside recursion.

For inherited $\mathsf{acc}_f = (\mathsf{com}_a, r_a, v_a)$ and new evaluations of p_f, a batched sumcheck reduces the different points to one fresh point z. After checking the terminal values a_z, p_z, a fresh challenge ρ gives $a'_f = a_f + \rho p_f$ and $\mathsf{acc}'_f = (\mathsf{com}_a + \rho\mathsf{com}_p, z, a_z + \rho p_z)$. Protocol 2 checks this homomorphic update and its challenge order, retaining one claim per family. Every family starts from the zero polynomial; empty updates leave its claim unchanged. Prefill and decode therefore share a fixed registry, even when their operators access different families.

For fixed parameters and tables, $a_f = s_f p_f$, so accumulation updates only $s_f \leftarrow s_f + \rho$; changing activation polynomials use dense updates. Separate family domains avoid padding every tensor to the largest shape. Mode-specific verifier classes $\phi_k \in \{\mathsf{Prefill}, \mathsf{Decode}\}$ exclude decode-only obligations from prefill. The new verifier instance u_k is folded in the next update or at closing.

3.4 End-to-End Protocol

Protocol 1 exposes Init, AppendChunk, and Finalize. Initialization fixes context ζ (model, numerical program, and label) and setup; each append binds a completed chunk, and finalization proves the accumulated prefix. *Sequence binding* connects these updates. For decode input $t_{k,i}$ and output $o_{k,i}$,

we check the registered greedy rule and tie-break, then enforce $t_{k,i+1} = o_{k,i}$ within a chunk and $t_{k+1,0} = o_{k,C-1}$ across decode chunks. The first decode input equals the last prompt token. These equalities and state commitments link every generation step to the same model and causal input history.

A running Poseidon state $h_k = \mathsf{Absorb}(h_{k-1}; \chi_k, \kappa_k, \mathbf{t}_k, o_{k,C-1})$ binds this execution to the public token sequence (Grassi et al., 2021). Here χ_k is the chained script digest; the final emission is included only in decode. Absorption uses the embedding and decoding variables themselves. The client reconstructs h_{K-1} from the public tokens and checks the final state and length.

Protocol 1 End-to-end incremental inference

Require: Authenticated context ζ and setup pp; live state $\mathcal{S} = (k, \sigma, u)$ and private witnesses.

procedure $\mathsf{Init}(\zeta; \mathsf{pp})$ ▷ Provider $\mathcal{P}$
  return $(0, \mathsf{Genesis}(\zeta; \mathsf{pp}), \bot)$
procedure $\mathsf{AppendChunk}(\mathcal{S}, x_k, w_k)$ ▷ Provider $\mathcal{P}$
  $\phi_k \leftarrow \mathsf{Class}(x_k)$; $\pi_k \leftarrow \mathsf{ProveChunk}(x_k, w_k; \sigma)$
  $\mathcal{E}_k \leftarrow \mathsf{Reduce}_{\phi_k}(x_k, \pi_k; \sigma)$
  $\mu_k \leftarrow \mathsf{PrepareUpdate}(\sigma, \mathcal{E}_k, u)$ ▷ Protocol 2
  $(\sigma, u) \leftarrow \mathsf{ProveStep}_{\phi_k}(\sigma, x_k, \pi_k, \mu_k, u)$ ▷ Protocol 3
  return $(k+1, \sigma, u)$
procedure $\mathsf{Finalize}(\mathcal{S})$ ▷ Provider $\mathcal{P}$
  $x^{(k)} \leftarrow \mathsf{Statement}(\zeta, \mathsf{scr}_{<k})$; $(\sigma_f, \mathbf{U}_f) \leftarrow \mathsf{Close}(\sigma, u)$
  $\pi_o \leftarrow \mathsf{OpenFamilies}(\sigma_f)$; $\pi_d \leftarrow \mathsf{ProveDeciders}(\mathbf{U}_f)$
  $\Pi \leftarrow (\sigma_f, \mathbf{U}_f, \pi_o, \pi_d)$; **return** $(x^{(k)}, \Pi)$
procedure $\mathsf{Verify}(x^{(k)}, \Pi; \zeta, \mathsf{pp})$ ▷ Client $\mathcal{V}$
  assert $\mathsf{Registered}(\zeta, \mathsf{pp})$ and $\mathsf{ValidScript}(x^{(k)})$
  assert $x^{(k)}$ matches the client's request
  return $\mathsf{VerifyTail}(x^{(k)}, \Pi)$

Private witnesses start neutral and persist across appends. VerifyTail checks token replay, final state, openings, and deciders; Appendix B.3.1 specifies terminal padding.

Each append carries accumulated evaluations and a pending verifier instance; closing folds the last pending primary and companion instances. Final openings settle the evaluations, while deciders and token replay bind the verifier chain to the public prefix. Appendix B.1 specifies prefix finalization; Theorem 5 gives the composition argument.

3.5 Complexity Analysis

A native chunk costs $W_{\mathsf{nat}}(C)$ for witness generation, GKR/lookup reductions, commitments, and accumulation; $W_{\mathsf{rec}}(C)$ covers its recursive update. At a fixed prefill/decode ratio, with finalization cost $W_{\mathsf{fin}}(C)$,

$$W_{\mathsf{prove}}(T, C) = \frac{T}{C}\big(W_{\mathsf{nat}}(C) + W_{\mathsf{rec}}(C)\big) + W_{\mathsf{fin}}(C) + \Theta(T). \tag{7}$$

At fixed C, longer executions repeat the same chunk relation and amortize finalization as $W_{\mathsf{fin}}(C)/T$ per token. Chunk width changes both the number of updates and the local work.

For hidden width d, N_H heads, and vocabulary V, dense witness computation costs $\Theta(N_L T(d^2 + dd_s) + N_L TC(N_H d_s + d) + T_{\mathsf{dec}} dV)$. The terms cover dense layers/boundaries, SSD, and the decode head; T_{dec} counts outputs and $N_H d_v = \Theta(d)$.

Accumulating n evaluations in a domain of size $N_f = 2^{d_f}$ takes $O(nN_f)$ prover and $O((n+1)d_f)$ verifier field work. Dense SSD uses $\Theta(C^2)$ entries per head and $\Theta(TC)$ across K chunks, trading larger local tensors for fewer updates. At fixed model, width, and security parameters, proving is $\Theta(T)$; terminal cryptographic verification and cryptographic proof size are $\Theta(1)$ in K. Public-token replay takes $\Theta(T)$ without rerunning inference. Appendix C.5 expands the model-size dependence and separates append costs from repeated prefix settlement and cold setup.

4 EXPERIMENTS

4.1 SETUP

We implement MPROOF in Rust for a pretrained 12-layer, 187M-parameter Mamba-3 SISO model. Proof experiments use one Google Cloud c4d-highmem machine with 48 cores and default chunk width $C = 128$. Equal prefill/decode chunk counts expose both phases; T counts their combined processed tokens. We run each performance configuration three times and report the mean. After model and setup preparation, prover time includes witness generation through final verification. The prepared verifier checks public tokens, final state, openings, and recursive deciders. Numerical evaluations compare BF16 and ChunkFP on the same checkpoint and tokenizer. Appendix A records the measurement and scoring protocols.

4.2 END-TO-END PERFORMANCE

Table 2: End-to-end performance on the 187M model. Proof size counts scalar/group elements at 32 bytes each; gray marks the default width at 1,024 tokens.

Total	Chunk size	Chunks	Prover	Avg. prover	Verifier	Proof size
tokens	*tokens*		*min*	*s/token*	*s*	*KB*
1,024	8	128	38.34	2.246	0.379	260.4
1,024	32	32	13.77	0.807	0.385	259.5
1,024	128	8	6.50	0.381	0.411	263.3
1,024	256	4	5.15	0.302	0.447	286.3
4,096	128	32	22.85	0.335	0.451	263.3
16,384	128	128	88.18	0.323	0.495	263.3

At $C = 128$, MPROOF proves 4,096 tokens in 22.85 minutes and 16,384 tokens in 88.18 minutes (Table 2). Across a $16\times$ increase in sequence length, total prover time grows by $13.6\times$ and average cost falls by 15.2%, to 0.323 seconds per token. Longer generations add bounded chunks rather than enlarge the model relation or recursive circuit.

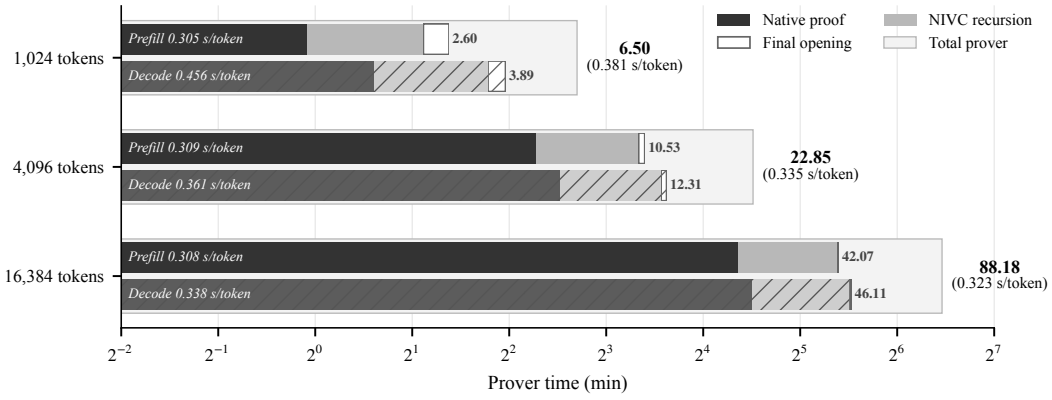

Figure 2: Sequence scaling at $C = 128$, separating native proving, recursion, and final opening for prefill and decode.

Figure 2 explains this scaling: prefill cost stays near 0.31 seconds per token, while decode amortizes finalization across more tokens. The two successive length intervals add almost the same time per extra token, matching Eq. (7). Prepared verification stays below 0.5 seconds, and the serialized cryptographic proof remains 248.4 KB. Accumulation and recursion thus compress longer executions into a fixed cryptographic verification problem, keeping the final proof compact as generation extends.

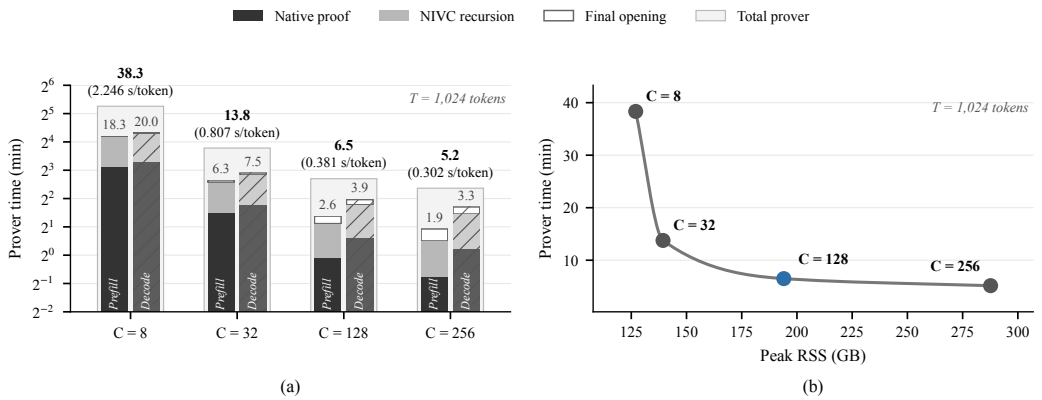

Figure 3: Chunk-width tradeoff at $T = 1{,}024$: (a) prover-time breakdown; (b) prover time versus peak RSS.

Chunk width makes proof efficiency tunable. At $T = 1{,}024$, increasing C from 8 to 256 gives a $7.44\times$ speedup with $32\times$ fewer recursive updates (Figure 3). Wider chunks share more proof work and amortize recursion, while their $\Theta(C^2)$ SSD tensors explain the diminishing returns. Moving from 128 to 256 saves another 20.7% of prover time as peak RSS rises from 194.1 to 287.7 GB. We use $C = 128$ to balance throughput and memory; the utility evaluation compares all four rounding programs.

At fixed $C = 128$, a $16\times$ longer sequence raises peak RSS by only 3.83%, from 194.1 to 201.5 GB. The prover retains the current chunk, accumulated polynomials, and recursive witnesses instead of the activation history. These results establish the architectural advantage of chunkwise proving: compared with token-level recursion, it shares proof work and amortizes composition; compared with transcript-wide proving, it keeps local instances bounded as generation extends. Chunk width therefore controls local resource demand, while sequence length determines the number of updates. Distributed proving could further reduce per-machine memory and wall-clock prover time (Wu et al., 2018).

4.3 Ablations

We disable token-wide operator sharing or SSD target sharing while preserving the numerical program, tokens, outputs, boundaries, and four recursive updates. Table 3 isolates their contribution to the same execution. Token sharing accelerates end-to-end proving by $1.79\times$ and GKR by $3.54\times$. It also cuts peak RSS from 235.8 to 124.4 GB, a **47.2% reduction in memory**.

Table 3: Sharing ablations at $C = 8, T = 32$. Gray is full MPROOF.

Shared checks		Prover	GKR	Peak RSS
Token	SSD	*s*	*s*	*GB*
✓	✓	**102.2**	**22.7**	**124.4**
×	✓	183.2	80.5	235.8
✓	×	107.0	28.1	132.2

Wide arithmetization removes duplicated proof work at its source. Eight token-local sumchecks become one, and claim-merging calls fall from 6,532 to 230 ($28.4\times$): sharing operators eliminates intermediate claims and their downstream merging obligations. The shorter transcript also reduces the verifier computation checked by recursion. SSD target sharing complements this benefit within recurrent contractions. The Gram and source-contraction kernels use $8\times$ fewer field multiplications, and the four core SSD proof routines run $4.55\times$ faster. Full GKR improves by $1.24\times$, with identical SSD witness computation in both variants.

Across chunks, accumulation makes terminal openings depend on the fixed family registry. For 16,384 tokens, the reached-family census gives 2,688 batches with independent per-chunk settlement and 22 after accumulation—a $122.18\times$ reduction, with $92.86\times$ less cumulative polynomial input volume (Appendix A.6). These results establish gains from proof organization beyond the efficiency of the underlying SSM. Wide arithmetization makes the same numerical execution cheaper to prove; accumulation keeps the final opening workload fixed as the sequence grows. Together, they reduce per-update cost and keep finalization compact for long generation.

4.4 Model Utility

Table 4: Model utility at $C = 128$ with the same checkpoint and matched inputs.

		Perplexity ↓			*Agreement (%)* ↑		*JS* ↓
Corpus	Tokens	BF16	ChunkFP	Δ (%)	Top-1	Top-5	10^{-3}
WikiText-2	16,384	24.3161	24.3989	+0.34	94.24	99.945	3.397
C4	17,964	26.0576	26.1797	+0.47	94.68	99.983	1.332
		Accuracy (%) ↑			*Correct answers*		
Task	Questions	BF16	ChunkFP	Δ (pp)	BF16	ChunkFP	Δ
LAMBADA	128	30.47	29.69	−0.78	39	38	−1
HellaSwag	128	34.38	34.38	+0.00	44	44	0
PIQA	128	72.66	71.88	−0.78	93	92	−1
ARC-Challenge	128	25.78	26.56	+0.78	33	34	+1
MMLU-Pro	128	10.16	9.38	−0.78	13	12	−1
Overall	640	**34.69**	**34.38**	**−0.31**	222	220	−2

Top-1: argmax agreement; Top-5: ChunkFP argmax in the BF16 top-5; JS: mean Jensen–Shannon divergence in nats. Δ: ChunkFP minus BF16, relative for PPL. Overall pools the five 128-question tasks; MMLU-Pro is zero-shot.

ChunkFP closely preserves predictive behavior while providing the specified arithmetic required for execution proofs. At $C = 128$, perplexity rises by 0.34% on WikiText-2 and 0.47% on C4 (Merity et al., 2017; Raffel et al., 2020), with over 94% top-1 agreement and 99.9% top-5 containment. Across five downstream tasks with 128 scored questions each, accuracy differs by at most one correct answer per task (0.78 percentage points; Table 4).

Table 5: Chunk-width sensitivity on 16,384 WikiText-2 tokens.

	Perplexity ↓		*Agreement (%)* ↑		*JS* ↓
C	ChunkFP	Δ (%)	Top-1	Top-5	10^{-3}
8	24.4798	+0.67	93.67	99.927	4.428
32	24.4211	+0.43	94.12	99.969	3.301
128	24.3989	**+0.34**	94.24	99.945	3.397
256	24.4353	+0.49	93.22	99.915	3.986

BF16 PPL: 24.3161 at all widths. Gray marks $C = 128$.

Across widths, perplexity rises by 0.34–0.67%, with $C = 128$ giving the smallest change on this input (Table 5). Its WikiText-2 result is shared with Table 4. Overall, MPROOF achieves efficient incremental proving while closely preserving the pretrained model's language modeling quality and downstream task accuracy.

5 Related Work

Structure-aware neural proofs. Specialized systems exploit convolutions, matrix products, and nonlinear operations (Liu et al., 2021; Hao et al., 2024; Qu et al., 2026); NanoZK composes layer-wise proofs (Wang, 2026). MPROOF exploits repeated operators, SSD contractions, and rounded trajectories within bounded proof updates.

Recursive proof composition. Kaizen accumulates evaluations by type, and *Proofs for Deep Thought* composes GKR reductions (Abbaszadeh et al., 2024; Bünz & Chen, 2025). Accumulation, folding, and non-uniform recursion provide the tools (Bünz et al., 2020; Kothapalli et al., 2022; Kothapalli & Setty, 2022; 2024; 2023; Long, 2026). MPROOF specializes these tools for fixed model data and mode-specific verifiers, then binds the resulting proof to the full inference sequence.

6 Conclusion

MPROOF combines a fixed recurrent interface with shared chunk computation to make long generation incrementally verifiable. Wide arithmetization reduces repeated checks and proof depth; optimized recursion and sequence binding authenticate the complete execution. On a pretrained 187M Mamba-3 model, it achieves near-linear prover growth, nearly constant cryptographic verification, and fixed cryptographic proof size while retaining close BF16 agreement. The result is an execution-integrity layer for long-running language-model services, with chunk width controlling proof work and memory throughout generation.

AI USE STATEMENT

Generative AI tools assisted with literature discovery, manuscript organization, and language editing. The authors take responsibility for the final manuscript.

REFERENCES

Kasra Abbaszadeh, Christodoulos Pappas, Jonathan Katz, and Dimitrios Papadopoulos. Zero-knowledge proofs of training for deep neural networks. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*, pp. 4316–4330. ACM, 2024. doi: 10.1145/3658644.3670316. URL https://eprint.iacr.org/2024/162.

Anthropic. Effective context engineering for AI agents, 2025a. URL https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents.

Anthropic. A postmortem of three recent issues, September 2025b. URL https://www.anthropic.com/engineering/a-postmortem-of-three-recent-issues.

Thomas Attema, Serge Fehr, and Michael Klooß. Fiat–Shamir transformation of multi-round interactive proofs (extended version). *Journal of Cryptology*, 36(4):36, 2023. doi: 10.1007/s00145-023-09478-y.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7432–7439, 2020. doi: 10.1609/aaai.v34i05.6239. URL https://ojs.aaai.org/index.php/AAAI/article/view/6239.

Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orrú. Gemini: Elastic SNARKs for diverse environments. In *Advances in Cryptology – EUROCRYPT 2022*, pp. 427–457. Springer, 2022. doi: 10.1007/978-3-031-07085-3_15.

Benedikt Bünz and Jessica Chen. Proofs for Deep Thought: Accumulation for large memories and deterministic computations. In *Advances in Cryptology – ASIACRYPT 2024*, volume 15488 of *Lecture Notes in Computer Science*, pp. 269–301. Springer, 2025. doi: 10.1007/978-981-96-0935-2_9. URL https://eprint.iacr.org/2024/325. Extended version revised March 2025.

Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Recursive proof composition from accumulation schemes. In *Theory of Cryptography*, pp. 1–18. Springer, 2020. doi: 10.1007/978-3-030-64378-2_1.

Lingjiao Chen, Matei Zaharia, and James Zou. How is ChatGPT's behavior changing over time? arXiv preprint arXiv:2307.09009, 2023. URL https://arxiv.org/abs/2307.09009.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. arXiv preprint arXiv:1803.05457, 2018. URL https://arxiv.org/abs/1803.05457.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 10041–10071. PMLR, 2024. URL https://proceedings.mlr.press/v235/dao24a.html.

Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pp. 113–122. ACM, 2008. doi: 10.1145/1374376.1374396.

Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 519–535. USENIX Association, 2021. ISBN 978-1-939133-24-3. URL https://www.usenix.org/conference/usenixsecurity21/presentation/grassi.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *The First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

Ulrich Haböck. Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive, Paper 2022/1530, 2022. URL https://eprint.iacr.org/2022/1530.

Tsimur Hadeliya, Mohammad Ali Jauhar, Nidhi Sakpal, and Diogo Cruz. When refusals fail: Unstable safety mechanisms in long-context LLM agents. arXiv preprint arXiv:2512.02445, 2025.

URL https://arxiv.org/abs/2512.02445. Accepted at the AAAI 2026 TrustAgent Workshop.

Meng Hao, Hanxiao Chen, Hongwei Li, Chenkai Weng, Yuan Zhang, Haomiao Yang, and Tianwei Zhang. Scalable zero-knowledge proofs for non-linear functions in machine learning. In *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 3819–3836. USENIX Association, 2024. ISBN 978-1-939133-44-1. URL https://www.usenix.org/conference/usenixsecurity24/presentation/hao-meng-scalable.

IEEE 802.11 Working Group. IEEE P802.11 Artificial Intelligence Offload Study Group, 2026. URL https://www.ieee802.org/11/Reports/aio_update.htm.

Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology – ASIACRYPT 2010*, pp. 177–194. Springer, 2010. doi: 10.1007/978-3-642-17373-8_11.

Abhiram Kothapalli and Srinath Setty. SuperNova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive, Paper 2022/1758, 2022. URL https://eprint.iacr.org/2022/1758.

Abhiram Kothapalli and Srinath Setty. CycleFold: Folding-scheme-based recursive arguments over a cycle of elliptic curves. Cryptology ePrint Archive, Paper 2023/1192, 2023. URL https://eprint.iacr.org/2023/1192.

Abhiram Kothapalli and Srinath Setty. HyperNova: Recursive arguments for customizable constraint systems. In *Advances in Cryptology – CRYPTO 2024*, pp. 345–379. Springer, 2024. doi: 10.1007/978-3-031-68403-6_11. URL https://eprint.iacr.org/2023/573. Extended version, Definition 12 and Lemma 4.

Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In *Advances in Cryptology – CRYPTO 2022*, volume 13510 of *Lecture Notes in Computer Science*, pp. 359–388. Springer, 2022. doi: 10.1007/978-3-031-15985-5_13. URL https://eprint.iacr.org/2021/370. Extended version revised July 2024.

Aakash Lahoti, Kevin Y. Li, Berlin Chen, Caitlin Wang, Aviv Bick, J. Zico Kolter, Tri Dao, and Albert Gu. Mamba-3: Improved sequence modeling using state space principles. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=HwCvaJOiCj.

Tianyi Liu, Xiang Xie, and Yupeng Zhang. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2968–2985. ACM, 2021. doi: 10.1145/3460120.3485379.

Jieyi Long. Interstellar: Efficient GKR-based IVC scheme with privacy-preserving collaborative folding. In *Public-Key Cryptography – PKC 2026*, volume 16553 of *Lecture Notes in Computer Science*, pp. 159–196. Springer, 2026. doi: 10.1007/978-3-032-26737-5_6. URL https://doi.org/10.1007/978-3-032-26737-5_6.

Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992. doi: 10.1145/146585.146605.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Byj72udxe.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534. Association for Computational Linguistics, 2016. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144/.

Shahar Papini and Ulrich Haböck. Improving logarithmic derivative lookups using GKR. Cryptology ePrint Archive, Paper 2023/1284, 2023. URL https://eprint.iacr.org/2023/1284.

Wenjie Qu, Yijun Sun, Xuanming Liu, Tao Lu, Yanpei Guo, Kai Chen, and Jiaheng Zhang. zkGPT: An efficient non-interactive zero-knowledge proof framework for LLM inference. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 2045–2063. USENIX Association, 2025. ISBN 978-1-939133-52-6. URL https://www.usenix.org/conference/usenixsecurity25/presentation/qu-zkgpt.

Wenjie Qu, Yanpei Guo, Yue Ying, and Jiaheng Zhang. VerfCNN, optimal complexity zkSNARK for convolutional neural networks. In *2026 IEEE Symposium on Security and Privacy*, pp. 3037–3054.

IEEE, 2026. doi: 10.1109/SP63933.2026.00028. URL https://eprint.iacr.org/2025/2020.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL https://jmlr.org/papers/v21/20-074.html.

Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Advances in Cryptology – CRYPTO 2020*, pp. 704–737. Springer, 2020. doi: 10.1007/978-3-030-56877-1_25.

Haochen Sun, Jason Li, and Hongyang Zhang. zkLLM: Zero knowledge proofs for large language models. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*, pp. 4405–4419. ACM, 2024. doi: 10.1145/3658644.3670334.

Florian Tramèr and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1806.03287.

Lizheng Wang, Hancheng Lou, Chongrong Li, Yu Yu, and Yuncong Hu. zkAgent: Verifiable LLM agent execution via one-shot transcript proofs. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2026. URL https://eprint.iacr.org/2026/199. To appear. Extended version: Cryptology ePrint Archive, Paper 2026/199.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-3018. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/ad236edc564f3e3156e1b2feafb99a24-Abstract.html.

Zhaohui Wang. NanoZK: Privacy-preserving verifiable inference for large language models via layerwise zero-knowledge proofs. arXiv preprint arXiv:2603.18046v2, 2026. URL https://arxiv.org/abs/2603.18046v2. Extended version of the ICICS 2026 paper.

Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. DIZK: A distributed zero knowledge proof system. In *27th USENIX Security Symposium (USENIX Security 18)*, pp. 675–692. USENIX Association, 2018. ISBN 978-1-939133-04-5. URL https://www.usenix.org/conference/usenixsecurity18/presentation/wu.

Jiashu Xu, Fei Wang, Mingyu Ma, Pang Wei Koh, Chaowei Xiao, and Muhao Chen. Instructional fingerprinting of large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3277–3306. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.naacl-long.180. URL https://aclanthology.org/2024.naacl-long.180/.

Fatemeh Zarinjouei, Maryam Zarezadeh, Behzad Abdolmaleki, Bhavish Mohee, Aysajan Abidin, and Stefan Köpsell. zkRNN: Zero-knowledge proofs for recurrent neural network inference. Cryptology ePrint Archive, Paper 2026/073, 2026. URL https://eprint.iacr.org/2026/073.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800. Association for Computational Linguistics, 2019. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pp. 159–177. ACM, 2021. doi: 10.1145/3460120.3484767.

Xiaoyuan Zhu, Yaowen Ye, Tianyi Qiu, Hanlin Zhu, Sijun Tan, Ajraf Mannan, Jonathan Michala, Raluca Ada Popa, and Willie Neiswanger. Auditing black-box LLM APIs with a rank-based uniformity test. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=PSIe9mmF7a.

A Experimental Details

A.1 Model and Corpus Evaluation

All evaluations use the 12-layer `state-spaces/mamba3-siso-187m` checkpoint. Proof experiments run three times per configuration on a c4d-highmem-96 machine (48 cores, approximately 787 GB available memory) with an admitted powers-of-tau SRS; reported times are means. Numerical evaluations run locally. State/value widths are $(d_s, d_v) = (128, 64)$; model and padded proof dimensions are

$$(d, N_H, d_{\mathsf{MLP}}, V) : (768, 24, 1{,}536, 128{,}256) \mapsto (1{,}024, 32, 2{,}048, 131{,}072).$$

Token identifiers satisfy $\mathsf{id} < V_{\mathsf{real}}$. ChunkFP uses a base 16-bit, 12-fractional-bit profile with operator-specific scales, ranges, tables, and rescale sites fixed by spec.

Table 4 scores all 16,384 requested WikiText-2 tokens (Merity et al., 2017) and 17,964 C4 tokens (Raffel et al., 2020) at $C = 128$. Let p_i, q_i be the BF16 and ChunkFP predictive distributions on the same prefix, y_i the target, and $\hat{y}_i^p, \hat{y}_i^q$ their canonical argmax tokens. For N scored tokens,

$$\mathrm{PPL}(p) = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log p_i(y_i)\right), \qquad \Delta_{\mathrm{PPL}} = 100\left(\frac{\mathrm{PPL}(q)}{\mathrm{PPL}(p)} - 1\right),$$

$$\mathrm{Top1} = \frac{100}{N}\sum_i [\hat{y}_i^q = \hat{y}_i^p], \qquad \mathrm{Top5} = \frac{100}{N}\sum_i [\hat{y}_i^q \in \mathrm{Top}_5(p_i)],$$

$$\mathrm{JS} = \frac{1}{2N}\sum_i \big(\mathrm{KL}(p_i\|m_i) + \mathrm{KL}(q_i\|m_i)\big), \qquad m_i = (p_i + q_i)/2.$$

JS uses natural logarithms and is displayed in units of 10^{-3}. The prefill relation audit checks chunk indices $\{0, 1, 15, 16, 127\}$ on WikiText-2 and $\{0, 1, 15, 16, 139\}$ on C4 against the prover's rounded relations. Performance runs generate complete proofs; numerical runs directly execute and score ChunkFP.

A.2 Fidelity across Chunk Widths

Table 5 scores the same 16,384 WikiText-2 tokens at $C \in \{8, 32, 128, 256\}$, using the metrics above and BF16 reference perplexity 24.3161094. The $C = 128$ measurements are shared with Table 4. Each width fixes its own ChunkFP rounding schedule while holding the checkpoint and input constant.

A.3 Downstream Scoring

Table 4 scores 128 questions each from LAMBADA (Paperno et al., 2016), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), ARC-Challenge (Clark et al., 2018), and MMLU-Pro (Wang et al., 2024); MMLU-Pro is zero-shot. For model M and task j, let $c_{M,j}$ count correct answers. The reported percentages and pooled score are

$$\mathrm{Acc}_{M,j} = \frac{100 c_{M,j}}{128}, \qquad \mathrm{Acc}_{M,\mathsf{all}} = \frac{100\sum_{j=1}^{5} c_{M,j}}{640} = \frac{1}{5}\sum_{j=1}^{5}\mathrm{Acc}_{M,j}.$$

Changes are ChunkFP minus BF16 in percentage points; one answer corresponds to $100/128 = 0.78125$ points. The Overall row pools the five tasks rather than introducing another evaluation.

A.4 Timing and Proof-Size Measurements

Prover timing excludes checkpoint/SRS loading and setup preparation. It includes witness generation, native proving and replay, accumulation, verifier folding, closing, final openings, and final verification. Admission precedes timing. Sizes use decimal units; peak RSS is the process high-water mark.

At $C = 128, T = 4{,}096$, public-token replay takes 0.022347 of the prepared verifier's 0.450716 seconds. For $T = (1{,}024, 4{,}096, 16{,}384)$, separate fresh-process verification takes $(0.542, 0.564, 0.634)$

seconds after setup, including bundle decoding and script validation. Full cold lifecycles, including SRS loading, setup, and teardown, take $(191.55, 215.86, 315.79)$ seconds.

At $C = 128$, the logical census is $32 \cdot 8{,}227 = 263{,}264$ bytes and the canonical cryptographic payload is $B_{\mathsf{crypto}} = 248{,}431$ bytes at every length. Each bundle has $B_{\mathsf{bundle}} = B_{\mathsf{crypto}} + B_{\mathsf{public}} + 60$ bytes. In the same length order,

$$\mathbf{B}_{\mathsf{public}} = (22{,}841, 60{,}041, 208{,}841),$$
$$\mathbf{B}_{\mathsf{bundle}} = (271{,}332, 308{,}532, 457{,}332) \text{ bytes.}$$

The 60 bytes are framing. Run records bind the checkpoint, program, SRS, source revision, and configuration; checkpoint and SRS payloads are separate reproduction inputs.

A.5 Memory and Execution Stages

At $T = 1{,}024$, widths $(8, 32, 128, 256)$ give peak RSS $(127.0, 139.4, 194.1, 287.7)$ GB. At $C = 128$, the length sweep gives $(194.1, 199.5, 201.5)$ GB, a 3.83% increase over $16\times$ more tokens. Figure 4 shows the per-chunk high-water marks.

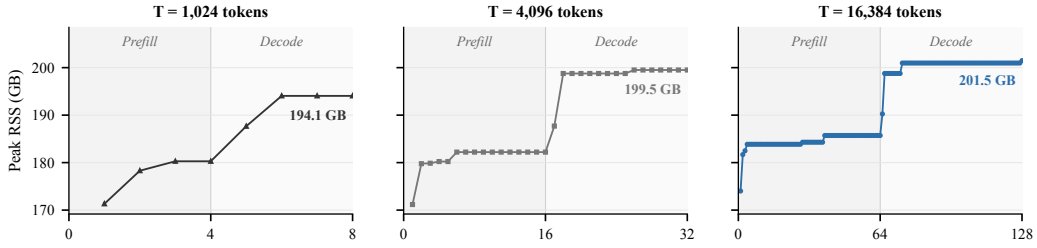

Figure 4: Per-chunk peak RSS at $C = 128$; shading separates prefill and decode.

Per layer, the recurrent representation sizes satisfy

$$M_{\mathsf{trajectory}} = \Theta(CN_H d_s d_v), \qquad M_{\mathsf{SSD}} = \Theta\big(N_H(C^2 + d_s d_v)\big)\,,$$
$$\frac{M_{\mathsf{SSD}}}{M_{\mathsf{trajectory}}} = \Theta\left(\frac{C}{d_s d_v} + \frac{1}{C}\right).$$

The SSD terms count source–target entries and the boundary; inputs, phases, tensor intermediates, and commitment material are additional. Accumulator polynomials and recursive witnesses persist across updates, while native transcripts can be released. At fixed model, width, and registry, retained proof storage is independent of completed chunk count; public-script storage and allocator behavior also affect RSS.

Figure 5 uses the archived labels "Token-split" and "Serial SSD" for disabled token-wide operator sharing and disabled SSD target sharing, respectively. Both retain the same ChunkFP execution as full MPROOF; Table 3 reports their end-to-end costs.

A direct-model-fold diagnostic on the full checkpoint at $C = 2, T = 8$ stops at its 717.5 GB safety limit during terminal GKR, before producing a proof.

A.6 Accounting for Shared Proof Work

Counts and timings come from the controlled proof traces; arithmetic counts are reconstructed from kernel loops and observed dimensions. Token sharing changes token-local sumchecks from 11,584 to 1,448, merge calls from 6,532 to 230, and GKR time from 80.46 to 22.72 seconds. Merge rounds decrease $68{,}890 \to 3{,}292$ $(20.93\times)$, index-pair visits $62{,}457{,}068 \to 7{,}503{,}642$ $(8.32\times)$, and selected projection-kernel multiplications $12{,}639{,}168 \to 4{,}429{,}032$ $(2.85\times)$.

Separate/shared SSD targets require $30{,}067{,}200/3{,}758{,}400 = 8$ times as many Gram/source-contraction proof multiplications. Mask composition, Gram contraction, mask application, and source contraction together take $2.364/0.520 \approx 4.55$ times as long. Both variants perform the same 19,156,992 witness multiplications for these formulas.

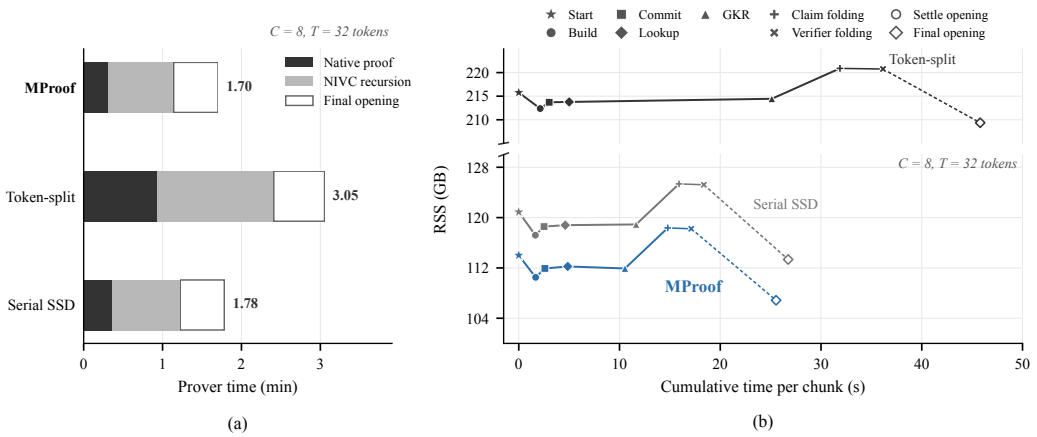

Figure 5: Stage-level view of the $C = 8, T = 32$ ablations: (a) prover-time composition; (b) memory at execution stages, in the original figure's GiB units.

At $C = 128, T = 16{,}384$, 64 prefill and 64 decode chunks reach 20 and 22 opening-width groups, respectively. With identical width batching,

$$\frac{B_{\mathsf{independent}}}{B_{\mathsf{accumulated}}} = \frac{64 \cdot 20 + 64 \cdot 22}{22} \approx 122.18, \qquad \frac{N_{\mathsf{independent}}}{N_{\mathsf{accumulated}}} = \frac{62{,}458{,}445{,}568}{672{,}641{,}532} \approx 92.86,$$

where B counts opening batches and N their input-table field elements. The native-transcript/final-proof element ratio is $11{,}366{,}703/8{,}227 \approx 1{,}381.63$.

At $C = 128, T = 4{,}096$, fixed-source accumulation uses 5,264 scalar additions instead of an estimated 9,573,638,144 dense coefficient updates. Packing changes fixed-source commitment counts $977 \to 17$ per prefill chunk and $984 \to 18$ per decode chunk, with padded coefficient-volume factors 1.289 and 1.254.

B Additional Method Details

B.1 Protocol Composition

Fix authenticated context $\zeta = (\mathsf{spec}, \mathsf{com}_\theta, \mathsf{com}_{\mathsf{tab}}, g, \mathsf{id}_{\mathrm{srs}})$. After k appends, Protocol 1 has live state and public prefix statement

$$\mathcal{S}_k = (k, \sigma_{k-1}, u_{k-1}; (a_{k-1,f})_{f \in \mathfrak{F}}, W_{k-1}^{\mathsf{fold}}), \qquad x^{(k)} = \mathsf{Statement}(\zeta, \mathsf{scr}_{<k}),$$

where $a_{k-1,f}$ are private accumulator polynomials and W_{k-1}^{fold} contains primary/companion folding witnesses. Genesis supplies the neutral entries at $k = 0$. Each update uses only $(\mathcal{S}_k, x_k, w_k)$; the client matches $x^{(k)}$ to its requested model and prompt.

Setup fixes Prefill, Decode, Wrap, and the family registry. A class digest binds operations, indices, constants, and transcript order while abstracting token/proof values. Each append must match that digest. The released batch runner prepares a complete script and finalizes once; its $C = 128$, $K \in \{8, 32, 128\}$ runs share the same setup digest.

The invariant after chunk k consists of the verifier chain represented by $(\{\mathcal{U}_{\phi,k}\}_\phi, u_k)$, its authenticated state/token record, and the family evaluation obligations. Protocol 3 checks the chain and record; Protocol 2 updates the obligations. Closing folds all pending primary/companion instances; final openings and deciders settle both against the same state (Theorems 4–5).

For live prefixes and finalization, respectively,

$$\mathsf{scr}_{<k} \in \{\varepsilon\} \cup \mathsf{Prefill}^+\mathsf{Decode}^*, \qquad \mathsf{scr}_{<k} \in \mathsf{Prefill}^+\mathsf{Decode}^+.$$

Here ε is the empty script. Continued appends use a retained unclosed state. Each delivered prefix proof repeats closing, openings, and deciders; Appendix C.5 accounts for these costs.

Protocol 2 Family accumulation: verifier checks

Require: Family f, accumulator $\mathsf{acc}_f = (\mathsf{com}_a, \mathbf{r}_a, v_a)$, claims $\mathcal{E}_f = (\mathsf{com}_p, \mathbf{r}_j, v_j)_{j\in[n]}$, proof μ_f, transcript τ.

procedure $\mathsf{Accumulate}.V(f, \mathsf{acc}_f, \mathcal{E}_f, \mu_f; \tau)$
  assert registered family sources and domains
  $\tau \leftarrow \mathsf{Absorb}(\tau; f, \mathsf{acc}_f, \mathcal{E}_f)$
  if $n = 0$ **then**
    return (acc_f, τ)
  end if
  $\xi \leftarrow \mathsf{Challenge}(\tau)$; $c_0 \leftarrow v_a + \sum_{j=0}^{n-1} \xi^{j+1} v_j$
  for $\ell = 1, \ldots, d_f$ **do**
    $g_\ell \leftarrow \mathsf{Read}(\mu_f)$
    assert $\deg g_\ell \le 2$ and $g_\ell(0) + g_\ell(1) = c_{\ell-1}$
    $z_\ell \leftarrow \mathsf{FS}(\tau; g_\ell)$; $c_\ell \leftarrow g_\ell(z_\ell)$
  end for
  $(a_z, p_z) \leftarrow \mathsf{Read}(\mu_f)$; for fixed f, read p_z and set $a_z = s_a p_z$
  assert Eq. (18); $\rho \leftarrow \mathsf{FS}(\tau; a_z, p_z)$
  $\mathsf{acc}'_f \leftarrow (\mathsf{com}_a + \rho\mathsf{com}_p, \mathbf{z}, a_z + \rho p_z)$
  For fixed f, update its scale: $s_a \leftarrow s_a + \rho$
  return (acc'_f, τ)
end procedure
Prover companion: generate μ_f by sumcheck and update $a_f \leftarrow a_f + \rho p_f$.

Protocol 3 Recursive update verifier

Require: Authenticated setup; prior state σ; chunk statement x_k, native proof π_k, update messages μ_k, pending instance u.
Failed checks return 0; $\mu_{k,f}$ is the accumulation transcript for family f.

procedure CHECKUPDATE$(\sigma, x_k, \pi_k, \mu_k, u)$
  assert fixed context ζ, class program digest, prefix grammar, and step index
  if $k = 0$ **then**
    assert $\sigma = \mathsf{Genesis}(\zeta; \mathsf{pp})$ and $u = \perp$
  else
    assert the predecessor instance and its bound state digest against σ
  end if
  assert incoming model boundary and prompt/decode token links
  $\phi \leftarrow \mathsf{Class}(x_k)$; $(\mathcal{E}, \tau) \leftarrow \mathsf{Reduce}_\phi(x_k, \pi_k; \sigma)$
  for each family f in the fixed registry order **do**
    $(\mathsf{acc}'_f, \tau) \leftarrow \mathsf{Accumulate}.V(f, \mathsf{acc}_f, \mathcal{E}_f, \mu_{k,f}; \tau)$ ▷ Protocol 2
  end for
  Update the statement record using the checked token variables in Eq. (20)
  if $u \neq \perp$ **then**
    assert the primary fold of u into its class accumulator using μ_k
  end if
  assert required companion folds, zero fresh error, and delegated commitment operands/outputs
  Form σ' from the checked boundary, token record, family claims, and folded instances
  Bind the canonical outgoing state digest
  return $(1, \sigma')$
end procedure

B.2 NOTATION

Bold symbols denote tensors; $\|$ denotes concatenation, and $[P]$ is the indicator of predicate P. Model state $\mathbf{s}_k$ enters chunk k; proof state σ_k follows its verification.

B.3 CHUNK INPUTS, STATE, AND DECODING

Use zero-based indices $[n] = \{0, \ldots, n-1\}$. The public statement is

$$x = (\mathsf{spec}, \mathsf{com}_\theta, \mathsf{com}_{\mathsf{tab}}, \mathsf{scr}, g, \mathsf{id}_{\mathrm{srs}}). \tag{8}$$

Table 6: Core notations.

Symbol	Description
Model and execution	
$T,\ C,\ K = T/C$	Token count; chunk width; chunk count
$N_\theta,\ N_L,\ d$	Parameter count; layer count; model width
$N_H,\ d_s,\ d_v$	Head count; state dimension per head; value width per head
spec	Registered execution specification
$\theta,\ \mathcal{T}$	Model parameters; lookup tables
$\mathsf{com}_\theta,\ \mathsf{com}_{\mathsf{tab}}$	Parameter commitment; table commitment
scr	Public token script
$\zeta,\ x^{(k)}$	Fixed context; statement for k completed chunks
$\kappa_k,\ \mathbf{t}_k,\ \mathbf{o}_k$	Chunk mode; input tokens; output tokens
$\mathbf{s}_k,\ \mathsf{com}_k^{\mathsf{st}}$	Incoming model state; state commitment
$x_k,\ w_k$	Local statement; model witness
Reduction and accumulation	
$\pi_k,\ \mathcal{E}_k$	Native chunk proof; exported claims
$\mathfrak{F},\ f$	Ordered family registry; family identifier
$d_f,\ N_f = 2^{d_f}$	Coordinate count; domain size
$\mathsf{acc}_{k,f},\ \mathbf{a}_k$	Family accumulator; accumulator vector
Recursive verification	
$\phi_k,\ \mathcal{G}_{\phi_k}$	Verifier class; recursive verifier relation
$\sigma_k,\ \delta_k$	Verifier state after chunk k; state digest
$\tau_k^{\mathsf{prf}},\ \tau_k^{\mathsf{stmt}}$	Proof transcript state; statement hash state
$u_k,\ \mathcal{U}_{\phi,k}$	Pending instance; folded class instance
Π	Final trajectory proof

The full-chunk script has $K_{\mathsf{pre}} > 0$ prefill chunks followed by $K_{\mathsf{dec}} > 0$ decode chunks. Each consumes C tokens; decode also emits C tokens. With mode κ_k, inputs $\mathbf{t}_k$, and emissions $\mathbf{o}_k$,

$$\begin{aligned} \mathsf{Prompt} &= \mathbf{t}_0 \| \cdots \| \mathbf{t}_{K_{\mathsf{pre}}-1} \| t_{K_{\mathsf{pre}},0}, \\ \mathsf{Output} &= \mathbf{o}_{K_{\mathsf{pre}}} \| \cdots \| \mathbf{o}_{K-1}, \qquad K = K_{\mathsf{pre}} + K_{\mathsf{dec}}. \end{aligned} \tag{9}$$

Thus $|\mathsf{Prompt}| = K_{\mathsf{pre}}C + 1$, $|\mathsf{Output}| = K_{\mathsf{dec}}C$, and $T = KC$: the last prompt token is the first decode input.

The boundary packs all layers in the fixed order and padding specified by spec:

$$\mathbf{s}_k = (H_{k,\ell}, \Psi_{k,\ell}, B^-_{k,\ell}, X^-_{k,\ell})_{\ell < N_L}, \qquad \mathsf{com}_k^{\mathsf{st}} = \mathsf{Commit}(\mathsf{pack}_{\mathsf{spec}}(\mathbf{s}_k)), \qquad \mathbf{s}_0 = 0.$$

The same $\mathsf{com}_{k+1}^{\mathsf{st}}$ labels chunk k's output and chunk $k+1$'s input. Local checks cover Table 7, parameter consistency, state packing, and ranges; decode also enforces the token links of Section 3.4.

For every checked field equality $L = R$, registered operand ranges satisfy

$$[L_{\min} - R_{\max},\ L_{\max} - R_{\min}] \subset (-p, p). \tag{10}$$

Hence $L = R \pmod p$ implies $L = R$ over the integers. For chosen output o and logits z_j, greedy decoding checks

$$z_o - z_j - [j < o] \in [0, 2^{b_{\mathsf{arg}}}), \qquad j \in [V], \qquad o < V_{\mathsf{real}}. \tag{11}$$

This fixes $o = \min \arg\max_{j < V_{\mathsf{real}}} z_j$, with difference width b_{arg}. The profile offsets real logits by $-8 \cdot 2^{e_{\mathsf{logit}}}$, sets padded logits to $-2^{15} + 1$, and requires the selected real logit to meet that floor. Selected values and differences are range checked.

B.3.1 Terminal padding extension

For $N > 0$ outputs, define

$$K_{\mathsf{dec}} = \lceil N/C \rceil, \qquad r = N - (K_{\mathsf{dec}} - 1)C, \qquad a_i = [i < r].$$

A terminal verifier class binds (N, r, stop), a canonical padding label, and a closing flag. It returns the first r terminal outputs and fixes inactive token/auxiliary fields to their padding values. The measured full-chunk instance has $r = C$.

For each token-local residual R_i, enforce $a_i R_i = 0$; count only active lookup queries and mask source–target contributions by $a_t a_s$. Active targets retain the width-C contractions and rescale schedule. Causality gives

$$t < r,\ s \le t \implies s < r, \qquad t_{i+1} = o_i\ (i+1 < r), \qquad \mathsf{s}_{\mathsf{out}} = \mathsf{Export}_{r-1}.$$

Here Export_{r-1} comprises the state, phase, and preceding input pair at the last active target; padded slots are no-ops. Active checks include ranges and argmax.

The terminal hash absorbs $(N, r, \mathsf{stop}, \mathbf{t}, o_{r-1})$ with the script digest. For the complete output $(y_i)_{i<N}$, EOS termination requires $y_{N-1} = \mathsf{EOS}$ and $y_i \neq \mathsf{EOS}$ for $i < N-1$; budget termination requires $N = N_{\max}$. The combined policy stops at the first event. Padding adds $C\lceil N/C \rceil - N \in [0, C-1]$ slots, at most 127 for $C = 128$.

B.4 Rounded SSD Relations

The registered rescale relation is

$$z + 2^{s-1} = 2^s q + r, \quad 0 \le r < 2^s, \qquad q = \mathsf{RS}_s(z) = \left\lfloor \frac{z + 2^{s-1}}{2^s} \right\rfloor. \tag{12}$$

It rounds half ties toward $+\infty$. Since $\mathsf{RS}_1(\mathsf{RS}_1(1 \cdot 1) \cdot 3) = 2 \neq 1 = \mathsf{RS}_1(1 \cdot \mathsf{RS}_1(1 \cdot 3))$, spec fixes contraction, rescale, and writeback order. Expressions below and in Table 7 use that schedule; arrows denote pre-rescale dependencies.

Suppress the layer index and let $u_{\mathsf{fp}} = 2^{e_u}$ encode one. For head j and $0 \le s \le t < C$, mask and carry decays satisfy

$$\begin{aligned} \Lambda_{t,s,j} &= \mathsf{RS}_{e_u}(\alpha_{t,j} \Lambda_{t-1,s,j}) + [t = s] u_{\mathsf{fp}}, \\ d_{t,j} &= \mathsf{RS}_{e_u}(\alpha_{t,j} d_{t-1,j}), \\ \Omega_{t,s,j} &= \mathsf{RS}_{e_u}(\Lambda_{t,s,j} \gamma_{s,j} + [s < t] \Lambda_{t,s+1,j} \beta_{s+1,j}). \end{aligned} \tag{13}$$

Seeds are $\Lambda_{-1,s,j} = 0$ and $d_{-1,j} = u_{\mathsf{fp}}$; out-of-domain entries are zero. Shifted copies bind predecessors, and ranges apply to every scheduled intermediate, including subsequently masked entries. The Gram and source-contraction shifts in Eq. (5) are $q_g = 2e_b - e_g$ and $q_y = e_g + e_x - e_y$, where e_b, e_x, e_g, e_y encode rotated B/C, values, Gram entries, and outputs.

The response is $Y = Y^{\mathsf{intra}} + Y^{\mathsf{state}} + Y^{\mathsf{prev}}$, with carry readouts and boundary export

$$\begin{aligned} Y_t^{\mathsf{state}} &\leftarrow d_t(\widehat{C}_t^{\mathsf{T}} H^-), & Y_t^{\mathsf{prev}} &\leftarrow (\Lambda_{t,0}\beta_0)(\widehat{C}_t^{\mathsf{T}} B^-) X^-, \\ H^{\mathsf{out}} &= D_H + S^\gamma + S^\beta + S^{\mathsf{prev}}, & (B^{-,\mathsf{out}}, X^{-,\mathsf{out}}, \Psi^{\mathsf{out}}) &= (\widehat{B}_{C-1}, X_{C-1}, \Psi_{C-1}). \end{aligned} \tag{14}$$

D_H is the rounded retained state. S^γ, S^β separately contract $\widehat{B}_s \otimes X_s$ with scheduled coefficients derived from $\Lambda_{C-1,s}\gamma_s$ and $[s < C-1]\Lambda_{C-1,s+1}\beta_{s+1}$. S^{prev} uses $\Lambda_{C-1,0}\beta_0$ and $B^- \otimes X^-$.

B.5 Arithmetic and Lookup Reductions

For an m-coordinate tensor, $\widetilde{X}(r) = \sum_{b \in \{0,1\}^m} \widetilde{\mathsf{eq}}(r, b) X(b)$. Write $[X; r, v]$ for $\widetilde{X}(r) = v$ and $\langle A \rangle_r = \sum_b \widetilde{\mathsf{eq}}(r, b) A(b)$. A gadget $Y = G(X_1, \ldots, X_m)$ reduces its output claim by

$$[Y; r, v] \xrightarrow{\mathsf{SC}(F_{G,r}, v)} (z, c), \qquad c \overset{?}{=} F_{G,r}(z; v_1, \ldots, v_m), \tag{15}$$

exporting $[X_i; \pi_i(r, z), v_i]$ under registered index maps π_i. Table 7 uses affine (A), pointwise (P), contraction (C), and wiring (W) rules:

$$\begin{array}{c|l} \mathsf{A} & v = \widetilde{c}_0(r) + \sum_i c_i \widetilde{X}_i(r) \\ \mathsf{P} & v = \langle \sum_i c_i \prod_h \widetilde{X}_{ih} \rangle_r \\ \mathsf{C} & v = \langle \sum_u \prod_h \widetilde{X}_h(\pi_h(b, u)) \rangle_r \\ \mathsf{W} & v = \sum_u W(r, u) \widetilde{X}(u). \end{array}$$

Affine and simple wiring cases reduce directly; the others use sumcheck. Shared-tensor claims merge before backward reduction. Slices, shifts, embedding selection, and state packing use the same index-map interface.

Like the source contraction in Eq. (6), the Gram reduction retains the shared head inside the sum:

$$\widetilde{G}^{\mathsf{raw}}(r_t, r_s, r_j) = \sum_{a,j} \widetilde{\mathsf{eq}}(r_j, j)\widetilde{\widehat{C}}(r_t, j, a)\widetilde{\widehat{B}}(r_s, j, a). \tag{16}$$

Both factors receive the same head challenge; index wiring and rescale checks connect this raw relation to Eq. (5).

The lookup rule L_f applies LogUp to queries q_i, table entries τ_j, and multiplicities m_j (Haböck, 2022):

$$\sum_i \frac{1}{\beta + q_i} = \sum_j \frac{m_j}{\beta + \tau_j}, \qquad m_j \in \mathbb{Z}_{\geq 0}, \quad \sum_j m_j < p.$$

Function tuples are compressed before β; source tuples and multiplicities are fixed before their challenges. Fraction-tree GKR uses leaves $(1, \beta + q_i)$, $(-m_j, \beta + \tau_j)$ and merge rule

$$(n_1, d_1) \oplus (n_2, d_2) = (n_1 d_2 + n_2 d_1, d_1 d_2).$$

The root numerator is zero, and every transmitted inverse satisfies $d\, d^{-1} = 1$. GKR exports MLE claims on query, output, multiplicity, and table leaves (Papini & Haböck, 2023).

Table 7 follows forward execution order. TW denotes token-wide operator checks; CW denotes chunk-wide recurrent checks. Here $S(z) = z\mathcal{T}_{\mathsf{sigmoid}}(z)$, M_Ψ is the phase modulus, and $\mathcal{I}_{\mathsf{st}} = (d, \Lambda, \beta, \gamma, \widehat{B}, X, H^-, B^-, X^-)$. Decay auxiliaries are quotients, residues, and shifted copies; coefficient C and normalization factor ρ are row-local names.

Table 7: Mamba-3 computations and their backward claim reductions.

Gadget / scope	Forward computation	Backward claim reduction
Embedding TW	$R_0[t, a] = E[\mathrm{id}_t, a]$	$R_0 \xrightarrow{\mathsf{W}} E$
RMSNorm TW	$S_t = \sum_a R_{\ell,t,a}^2, \quad \rho = \mathcal{T}_{\mathsf{rsqrt}}(S)$ $U_\ell = \rho R_\ell$	$U_\ell \xrightarrow{\mathsf{P}} (\rho, R_\ell)$ $\rho \xrightarrow{\mathsf{L}} S \xrightarrow{\mathsf{C}} R_\ell$
Input projection TW	$(X, Z, B, C, p_\Delta, p_A, p_\lambda, p_\Theta)$ $= \mathsf{Project}(U_\ell; W_{\mathsf{in}})$	$(X, \ldots, p_\Theta) \xrightarrow{\mathsf{C,W}} (U_\ell, W_{\mathsf{in}})$
Coefficients TW	$\Delta = \mathcal{T}_{\mathsf{softplus}}(p_\Delta + b_\Delta)$ $A = \mathcal{T}_{\mathsf{heavy}}(p_A), \quad \lambda = \mathcal{T}_{\mathsf{sigmoid}}(p_\lambda)$	$\Delta \xrightarrow{\mathsf{L,A}} (p_\Delta, b_\Delta)$ $(A, \lambda) \xrightarrow{\mathsf{L}} (p_A, p_\lambda)$
Write weights TW	$\alpha = \mathcal{T}_{\mathsf{exp}}(-\Delta A)$ $v_\beta = (u_{\mathsf{fp}} - \lambda)\Delta$ $\beta = v_\beta \alpha, \quad \gamma = \lambda\Delta$	$\beta \xrightarrow{\mathsf{P}} (v_\beta, \alpha)$ $(v_\beta, \gamma) \xrightarrow{\mathsf{P,A}} (\lambda, \Delta)$ $\alpha \xrightarrow{\mathsf{L,P}} (\Delta, A)$
BCNorm TW	$\rho^V = \mathcal{T}_{\mathsf{rsqrt}}(\sum_a V_{t,a}^2)$ $V^b = g^V(\rho^V V) + b^V, \quad V \in \{B, C\}$	$V^b \xrightarrow{\mathsf{A,P}} (\rho^V, V, g^V, b^V)$ $\rho^V \xrightarrow{\mathsf{L,C}} V$
Phase step TW	$\Theta = \mathcal{T}_{\tanh \pi}(p_\Theta)$ $D = \Delta\Theta$	$D \xrightarrow{\mathsf{P}} (\Delta, \Theta)$ $\Theta \xrightarrow{\mathsf{L}} p_\Theta$
Phase scan CW	$P_t = \Psi^- + \sum_{s \leq t} D_s$ $P = M_\Psi q + \Psi$	$\Psi \xrightarrow{\mathsf{A}} (P, q)$ $P \xrightarrow{\mathsf{W}} (\Psi^-, D)$
Rotation TW	$(c, s) = (\mathcal{T}_{\cos}(\Psi), \mathcal{T}_{\sin}(\Psi))$ $(\widehat{v}_0, \widehat{v}_1) = (cv_0 - sv_1, sv_0 + cv_1)$	$(\widehat{B}, \widehat{C}) \xrightarrow{\mathsf{P,W}} (B^b, C^b, c, s)$ $(c, s) \xrightarrow{\mathsf{L}} \Psi$
Decay checks CW	(Λ, d) : Eq. (13)	$0 \xrightarrow{\mathsf{P,A,W}} (\alpha, \Lambda, d, \mathsf{aux})$

Continued on the next page.

Table 7 continued.

Gadget / scope	Forward computation	Backward claim reduction
SSD mask CW	$\Omega_{t,s} = \Lambda_{t,s}\gamma_s + [s < t]\Lambda_{t,s+1}\beta_{s+1}$	$\Omega \xrightarrow{\mathsf{P,W}} (\Lambda, \beta, \gamma)$
Gram product CW	$G_{t,s,j} = \sum_a \widehat{C}_{t,j,a}\widehat{B}_{s,j,a}$ $M = \Omega \odot G$	$M \xrightarrow{\mathsf{P}} (\Omega, G)$ $G \xrightarrow{\mathsf{C}} (\widehat{C}, \widehat{B})$
Source contraction CW	$Y^{\mathsf{intra}}_{t,j,e} = \sum_s M_{t,s,j}X_{s,j,e}$	$Y^{\mathsf{intra}} \xrightarrow{\mathsf{C}} (M, X)$
Carry readout CW	$Y^{\mathsf{state}}_t = d_t(\widehat{C}_t^{\mathsf{T}}H^-)$ $Y^{\mathsf{prev}}_t = (\Lambda_{t,0}\beta_0)(\widehat{C}_t^{\mathsf{T}}B^-)X^-$ $Y = Y^{\mathsf{intra}} + Y^{\mathsf{state}} + Y^{\mathsf{prev}}$	$Y \xrightarrow{\mathsf{A}} (Y^{\mathsf{intra}}, Y^{\mathsf{state}}, Y^{\mathsf{prev}})$ $Y^{\mathsf{state}} \xrightarrow{\mathsf{P,C}} (d, \widehat{C}, H^-)$ $Y^{\mathsf{prev}} \xrightarrow{\mathsf{P,C,W}} (\Lambda, \beta, \widehat{C}, B^-, X^-)$
State export CW	$H^{\mathsf{out}} = D_H + S^\gamma + S^\beta + S^{\mathsf{prev}}$ $B^{-,\mathsf{out}} = \widehat{B}_{C-1}, \quad X^{-,\mathsf{out}} = X_{C-1}$ $\Psi^{\mathsf{out}} = \Psi_{C-1}$	$H^{\mathsf{out}} \xrightarrow{\mathsf{A}} (D_H, S^\gamma, S^\beta, S^{\mathsf{prev}})$ $(D_H, S^\gamma, S^\beta, S^{\mathsf{prev}}) \xrightarrow{\mathsf{P,C,W}} \mathcal{I}_{\mathsf{st}}$ $(B^{-,\mathsf{out}}, X^{-,\mathsf{out}}, \Psi^{\mathsf{out}}) \xrightarrow{\mathsf{W}} (\widehat{B}, X, \Psi)$
Output gate TW	$Y^+ = Y + D_{\mathsf{skip}}X$ $F_{\mathsf{ssm}} = Y^+S(Z)$	$F_{\mathsf{ssm}} \xrightarrow{\mathsf{P,L}} (Y^+, Z)$ $Y^+ \xrightarrow{\mathsf{A,P}} (Y, D_{\mathsf{skip}}, X)$
SSM residual TW	$R'_\ell = F_{\mathsf{ssm}}W_o^{\mathsf{T}} + aR_\ell$	$R'_\ell \xrightarrow{\mathsf{C,A}} (F_{\mathsf{ssm}}, W_o, R_\ell)$
SwiGLU TW	$U'_\ell = \mathsf{RMSNorm}(R'_\ell)$ $G_m = U'_\ell W_g^{\mathsf{T}}, \quad V_m = U'_\ell W_u^{\mathsf{T}}$ $F_m = S(G_m)V_m$	$F_m \xrightarrow{\mathsf{P,L}} (G_m, V_m)$ $(G_m, V_m) \xrightarrow{\mathsf{C}} (U'_\ell, W_g, W_u)$ $U'_\ell \xrightarrow{\mathsf{RMSNorm}} R'_\ell$
FFN residual TW	$R_{\ell+1} = F_m W_d^{\mathsf{T}} + a' R'_\ell$	$R_{\ell+1} \xrightarrow{\mathsf{C,A}} (F_m, W_d, R'_\ell)$
LM head TW	$Z = U_{\mathsf{final}}E^{\mathsf{T}}, \quad z_o = \mathsf{Gather}(Z, o)$ $m_j = z_o - z_j - [j < o]$	$m \xrightarrow{\mathsf{A,W}} Z \xrightarrow{\mathsf{C}} (U_{\mathsf{final}}, E)$ $m \xrightarrow{\mathsf{L_{range}}} \mathcal{T}$ (decode)

RMSNorm maps $R_{\ell+1}$ to the next layer's input and produces U_{final} at the end. Rotation leaves unspecified coordinates unchanged; state exports follow Eq. (14) and pack into $\mathbf{s}_{k+1}$. Their three-factor contraction reduces through

$$\sum_{s,j} \widetilde{\mathsf{eq}}(r_j, j)\widetilde{c}(j, s)\widetilde{B}(r_a, j, s)\widetilde{X}(r_e, j, s),$$

where c is the scheduled carry coefficient.

B.6 Family Accumulation

The registry $\mathfrak{F}$ fixes each family's role, domain $N_f = 2^{d_f}$, and commitment source. After chunk k, $\mathbf{a}_k = (\mathsf{acc}_{k,f})_{f\in\mathfrak{F}}$. Initialize $a_{-1,f} = 0$ and $\mathsf{acc}_{-1,f} = (\mathsf{Commit}(0), \mathbf{0}, 0)$; empty updates copy the accumulator.

For a nonempty update, inherited $\mathsf{acc}_f = (\mathsf{com}_a, \mathbf{r}_a, v_a)$ asserts $\widetilde{a}(\mathbf{r}_a) = v_a$, and all claims $\mathcal{E}_f = ((\mathsf{com}_p, \mathbf{r}_j, v_j))_{j\in[n]}$ concern the same p_f. Fix these claims before sampling ξ and running

$$\sum_{\mathbf{b}\in\{0,1\}^{d_f}} \left(\widetilde{\mathsf{eq}}(\mathbf{b}, \mathbf{r}_a)\widetilde{a}(\mathbf{b}) + \sum_{j=0}^{n-1} \xi^{j+1}\widetilde{\mathsf{eq}}(\mathbf{b}, \mathbf{r}_j)\widetilde{p}(\mathbf{b}) \right) = v_a + \sum_{j=0}^{n-1} \xi^{j+1}v_j. \tag{17}$$

Each variable has degree at most two. After d_f rounds ending at $\mathbf{z}$, supplied values a_z, p_z must satisfy

$$c_{d_f} = \widetilde{\mathsf{eq}}(\mathbf{z}, \mathbf{r}_a)a_z + \left(\sum_{j=0}^{n-1} \xi^{j+1}\widetilde{\mathsf{eq}}(\mathbf{z}, \mathbf{r}_j) \right) p_z. \tag{18}$$

Only then sample ρ and set

$$a' = a + \rho p, \qquad \mathsf{acc}'_f = (\mathsf{com}_a + \rho \mathsf{com}_p,\ \mathbf{z},\ a_z + \rho p_z). \tag{19}$$

Protocol 2 checks these updates inside Protocol 3, absorbing each message before its domain-separated challenge. For fixed parameter/table families,

$$a_f = s_a p_f, \qquad \mathsf{com}_a = s_a \mathsf{com}_p, \qquad a_z = s_a p_z, \qquad s'_a = s_a + \rho.$$

Only p_z is transmitted and s_a updated; the registered base p_f, com_p remains fixed even at $s_a = 0$.

B.7 Recursive Verifier Interface

For $\phi_k = \mathsf{Class}(x_k)$, $\mathcal{G}_{\phi_k}$ implements Protocol 3. Its witness contains proof messages and verifier intermediates; accumulator polynomials and folding witnesses remain in the live prover state.

Write $h_k = \tau_k^{\mathsf{stmt}}$ for the Poseidon record. Keccak chains the canonical script encoding from seed $\chi_{-1} = H_{\mathsf{Kec}}(\mathsf{gen} \| g)$. With $\boldsymbol{\ell}_k$ the two 128-bit limbs of χ_k and y_k^{last} the last decode emission (zero in prefill),

$$\begin{aligned} \chi_k &= H_{\mathsf{Kec}}(\chi_{k-1} \| \mathsf{enc}_{\mathsf{scr}}(x_k)), \\ \tau_k^{\mathsf{stmt}} &= \mathsf{StmtAbsorb}(\tau_{k-1}^{\mathsf{stmt}}; \boldsymbol{\ell}_k, \kappa_k, \mathbf{t}_k, y_k^{\mathsf{last}}). \end{aligned} \tag{20}$$

Absorption uses the embedding/decoding token variables. Final replay matches both records to the public script. The digest $\delta_k = H_{\mathsf{state}}(\mathsf{enc}(\sigma_k))$ also binds setup, index, proof transcript τ_k^{prf}, outgoing boundary, last-token/mode fields, family claims, and primary/companion folded instances.

For recorded verifier program P_ϕ and emitted constraints $\mathcal{C}_\phi$, assume

$$\begin{aligned} \mathsf{NativeVerify}_\phi(x, \mu, \sigma) = (1, \sigma') &\iff \mathsf{Interpret}(P_\phi, x, \mu, \sigma) = (1, \sigma') \\ &\iff \exists \omega : \mathcal{C}_\phi(\sigma'; \sigma, x, \mu, \omega) = 1. \end{aligned} \tag{21}$$

Independent IR interpretation, structural/digest checks, and omission/mutation tests audit this correspondence. At $C = 128, T = 4{,}096$, the 32 chunk tapes contain 85,344 sumchecks with matching emitted relations, 9,376 direct checks, and 5,888 accumulator updates.

The primary relation has degree five over BN254, four committed cross terms, and commitments using the admitted MKZG powers. A degree-two Grumpkin CycleFold companion uses a separate Pedersen basis for delegated commitment operations (Kothapalli & Setty, 2023). Fresh instances set the homogenizing scalar to one and error to zero. Primary constraints bind companion operands, results, and canonical challenge decomposition. Neutral instances seed each class; Wrap folds all pending primary/companion work.

Finalization checks width-grouped MKZG/Gemini family openings, a sumcheck–MKZG primary decider, and a sumcheck–IPA companion decider (Kate et al., 2010; Bootle et al., 2022; Setty, 2020). Every reached class and Wrap is checked against the final state; unreached classes remain neutral.

C Theoretical Analysis

C.1 Equivalence of Wide and Positionwise Chunk Relations

Proof of Lemma 1. Let $\mathcal{R}_{\mathsf{wide}}, \mathcal{R}_{\mathsf{pos}}$ denote the two constraint systems, and let π_{sem} select model intermediates, outputs, and boundaries, discarding proof auxiliaries.

Repeated operators. Let $F_g(r_t, r_b)$ be the equality-weighted residual sum in Eq. (3). Boolean interpolation gives

$$F_g(\mathsf{bin}_c(t), b) = P_g(t, b), \qquad F_g \equiv 0 \iff \forall (t, b) \in [C] \times \{0,1\}^v :\ P_g(t, b) = 0.$$

The bijection $t \leftrightarrow \mathsf{bin}_c(t)$ therefore preserves satisfying assignments. Random evaluation of this identity is accounted for in Lemma 2.

Rounded trajectories. Let b_t be the registered seed addition (zero for carry decay). For each scalar trajectory coordinate, enforce

$$U_t^- = \begin{cases} U^{\mathsf{in}}, & t = 0, \\ U_{t-1}, & t > 0, \end{cases} \qquad U_t = \mathsf{RS}_{s_t}(\alpha_t U_t^-) + b_t =: F_t(U_t^-),$$

The integer ranges and Eq. (12) uniquely determine each quotient and remainder, so induction yields

$$U_t = (F_t \circ F_{t-1} \circ \cdots \circ F_0)(U^{\mathsf{in}}).$$

This is the positionwise trajectory, including the diagonal seeds and zero entries of Eq. (13). Conversely, that trajectory supplies every shifted copy and rescale witness.

SSD and composition. Equations (6) and (16) take a common linear evaluation over targets while keeping the shared head inside the product sum. Operands, causal weights, and rescale sites are unchanged, as are embedding, lookup, decoding, and boundary constraints. Composing the gadget equivalences gives

$$\{\pi_{\mathsf{sem}}(w) : \mathcal{R}_{\mathsf{wide}}(w) = 1\} = \{\pi_{\mathsf{sem}}(w) : \mathcal{R}_{\mathsf{pos}}(w) = 1\}. \qquad \square$$

C.2 Local Relations and Numerical Determinacy

Fix the registered model/table oracles, incoming state, and input tokens. Equation (10) lifts field equalities to $\mathbb{Z}$; Eq. (12) determines each rescale uniquely; functional lookup tables determine nonlinear outputs. Induction over operator dependencies and layers, followed by the tie-break in Eq. (11), gives

$$|\{\pi_{\mathsf{sem}}(w) : \mathcal{R}_{\mathsf{chunk}}(x_k; w) = 1\}| \le 1.$$

Here $\mathcal{R}_{\mathsf{chunk}}$ is the local ChunkFP relation. Every admitted execution supplies a satisfying witness.

Lemma 2 (Local reduction)**.** *For an admitted specification and immutable input-polynomial oracles, a native chunk verifier accepts a false chunk statement with all exported claims true with probability at most $\epsilon_k^{\mathsf{GKR}} + \epsilon_k^{\mathsf{lookup}}$, under fresh independent challenges.*

Proof. Outside the GKR/sumcheck and LogUp error events, true exported evaluations imply all gadget relations (Lund et al., 1992; Goldwasser et al., 2008; Zhang et al., 2021; Haböck, 2022; Papini & Haböck, 2023). Shared wiring and registered sources identify their operands. Numerical determinacy then gives a valid ChunkFP execution; a union bound yields the claim. $\square$

C.3 Accumulation Preserves Evaluation Obligations

Lemma 3 (Preservation of a false claim)**.** *With immutable polynomial oracles and fresh independent challenges, a nonempty Protocol 2 update accepts at least one false input claim but returns a true claim with probability at most $(n + 2d_f + 1)/|\mathbb{F}|$. These bounds add across adaptive updates.*

Proof. Fix the input errors before sampling ξ:

$$e_a = v_a - \widetilde{a}_f(\mathbf{r}_a), \qquad e_j = v_j - \widetilde{p}_f(\mathbf{r}_j), \qquad E(X) = e_a + \sum_{j=0}^{n-1} X^{j+1} e_j.$$

A false input gives $E \not\equiv 0$ and $\deg E \le n$, hence $\Pr[E(\xi) = 0] \le n/|\mathbb{F}|$. Otherwise Eq. (17) has a false target, which survives its d_f degree-two sumcheck rounds except with probability $2d_f/|\mathbb{F}|$.

After Eq. (18) is checked, at least one terminal error $e'_a = a_z - \widetilde{a}_f(\mathbf{z})$ or $e'_p = p_z - \widetilde{p}_f(\mathbf{z})$ is then nonzero. The output error is

$$e_{\mathsf{out}} = (a_z + \rho p_z) - (\widetilde{a}_f + \rho \widetilde{p}_f)(\mathbf{z}) = e'_a + \rho e'_p.$$

Both errors precede ρ, so $\Pr[e_{\mathsf{out}} = 0] \le 1/|\mathbb{F}|$. For fixed families, $e_{\mathsf{out}} = (s_a + \rho)e'_p$, including $s_a = 0$. Summing the three events gives the bound; conditioning on prior transcripts extends it to adaptive updates. Empty updates preserve their claim exactly. $\square$

C.4 From Local Relations to the End-to-End Statement

The relation $\mathcal{R}_{\mathsf{traj}}$ requires a valid script, canonical genesis, all local ChunkFP relations, and equal adjacent state commitments. The native verifier checks reductions, accumulation, state/token links, and exact terminal oracle evaluations.

Theorem 4 (Accumulated sequence reduction)**.** *For a fixed admitted context and independent challenges, the native verifier accepts a false $\mathcal{R}_{\mathsf{traj}}$ statement with probability at most*

$$\epsilon_{\mathsf{red}}(K) = \sum_{k<K} (\epsilon_k^{\mathsf{GKR}} + \epsilon_k^{\mathsf{lookup}}) + \sum_{k<K,f} \epsilon_{k,f}^{\mathsf{acc}}, \qquad \epsilon_{k,f}^{\mathsf{acc}} \le \frac{n_{k,f} + 2d_f + 1}{|\mathbb{F}|}, \tag{22}$$

with $\epsilon_{k,f}^{\mathsf{acc}} = 0$ for empty updates.

Proof. Outside the events of Lemmas 2 and 3, a false local relation produces a false terminal claim. Acceptance therefore gives valid local executions with common parameters and linked states. For each decode chunk, the checked links imply

$$\mathbf{o}_k = (t_{k,1}, \ldots, t_{k,C-1}, o_{k,C-1}).$$

The input blocks and final emissions thus determine the complete script. Their replay assembles $\mathcal{R}_{\mathsf{traj}}$; a conditional union bound over all chunks and families gives Eq. (22). □

Folding and recursion. Assume knowledge soundness and NIVC compatibility for the instantiated primary/companion backend (Kothapalli & Setty, 2024, Definition 12 and Lemma 4), with delegated operations handled by CycleFold (Kothapalli & Setty, 2023). For the degree-D homogeneous relation $\mathcal{H}_\phi$, fix input witnesses z_0, z_1, errors e_0, e_1, and mixed coefficients c_i before sampling η. The fold discrepancy is

$$\Delta(\eta) = \mathcal{H}_\phi(z_0 + \eta z_1) - (e_0 + \sum_{i=1}^{D-1} \eta^i c_i + \eta^D e_1). \tag{23}$$

Its extreme coefficients satisfy

$$[X^0]\Delta = \mathcal{H}_\phi(z_0) - e_0, \qquad [X^D]\Delta = \mathcal{H}_\phi(z_1) - e_1.$$

An invalid input makes some coordinate a nonzero polynomial of degree at most D. Thus a uniform challenge in S gives local error $D/|S|$, plus the sampler's statistical distance from uniform. For valid inputs, honest mixed coefficients give $\Delta \equiv 0$. The primary and companion degrees are five and two, respectively.

Fiat–Shamir. Challenges absorb the statement and preceding messages under separate protocol domains. Assume the committed native transcript is $\mathbf{k}$-out-of-$\mathbf{N}$ special-sound with interactive knowledge error $\kappa_{\mathsf{nat}}(K)$. Against Q random-oracle queries, Attema et al. (2023, Theorem 4) gives

$$\epsilon_{\mathsf{FS}}^{\mathsf{nat}}(K, Q) \le (Q+1)\kappa_{\mathsf{nat}}(K).$$

Theorem 5 (End-to-end composition)**.** *Fix an authenticated context and a finite execution. Assume Eq.* (21)*, binding commitments, collision-resistant hashes, and the recursive and Fiat–Shamir component premises above. Then Protocol 1 is a computationally sound argument for $\mathcal{R}_{\mathsf{traj}}$ in the random-oracle model.*

Proof. Recursive knowledge soundness and deciders recover witnesses for every reached class and Wrap. Equation (21) identifies their constraints with Protocol 3, including its native reductions and Protocol 2 calls. Closing covers all pending work; final openings bind the polynomial assignments; state/token hashes bind $x^{(K)}$. Therefore

$$\Pr[\mathsf{Verify}(x, \Pi) = 1 \wedge \nexists w : \mathcal{R}_{\mathsf{traj}}(x; w) = 1] \le \epsilon_{\mathsf{FS}}^{\mathsf{nat}} + \epsilon_{\mathsf{rec}} + \epsilon_{\mathsf{PCS}} + \epsilon_{\mathsf{hash}},$$

where the last three terms charge recursive extraction/decider failures, commitment/opening failures, and hash collisions. Outside these events, Theorem 4 gives the trajectory. The stated component premises bound each error. □

Algebraic error accounting. The release ledger counts independently challenged native checks, folds, terminal reductions, and SRS sampling. A sumcheck with m_j variables and degree δ_j contributes $m_j\delta_j/p$; root/merge tests are additional, and lookup fraction-tree sumchecks are counted once here. A lookup lane with a query and b table leaves adds

$$\frac{2(a+b)-1+s}{p}, \qquad s=\begin{cases} a+b, & \text{function-tuple compression,} \\ 0, & \text{range lookup.} \end{cases}$$

Accumulation adds n/p for input batching and $1/p$ for terminal cancellation. The artifact's `gamma` is Protocol 2's ξ.

For $C = 128$ and BN254 scalar-field order p,

$$\epsilon_{\mathsf{alg}}(K) = \frac{A_K}{p} + \frac{B_K}{2^{128}}, \qquad B_K = 22K + 134, \tag{24}$$

where A_K sums full-field reduction, lookup, accumulation, decider, opening, and SRS terms. The bounded-challenge charges are

$$B_K = \underbrace{10(K+1)}_{\text{primary folds}} + \underbrace{4(K+1)}_{\text{companion folds}} + \underbrace{4(2K+21)}_{\text{sampling}} + \underbrace{2\cdot 18}_{\text{IPA rounds}} .$$

Table 8: Algebraic error budgets under independent challenges at $C = 128$. Algebraic bits denote $-\log_2 \epsilon_{\mathsf{alg}}$.

Tokens	K	A_K	B_K	Algebraic bits
1,024	8	4,009,319,426	310	119.724
4,096	32	15,634,616,342	838	118.289
16,384	128	62,135,804,006	2,950	116.474

Table 8 gives $-\log_2 \epsilon_{\mathsf{alg}} \geq 116.474 > 112$, the registered target. Fiat–Shamir and computational losses are governed by Theorem 5.

C.5 Scaling in Model Size, Width, and Sequence Length

Let $N_L, d, V, T_{\mathsf{dec}}$ denote layers, hidden width, padded vocabulary, and decoded tokens. With fixed projection/MLP expansion, $N_H d_v = \Theta(d)$, and constant-factor padding, dense relation evaluation costs

$$W_{\mathsf{rel}}(T,C) = \Theta\big(N_L T(d^2 + dd_s) + N_L TC(N_H d_s + d) + T_{\mathsf{dec}} dV\big). \tag{25}$$

The three terms are dense layers/boundaries, SSD, and the decode head. At fixed model and decode fraction, only SSD grows with C: its $\Theta(TC)$ cost is quadratic for $C = T$ and linear for fixed C.

Using the phase-weighted native, recursive, and finalization costs of Section 3.5,

$$\frac{W_{\mathsf{prove}}(T,C)}{T} = \frac{W_{\mathsf{nat}}(C) + W_{\mathsf{rec}}(C)}{C} + \frac{W_{\mathsf{fin}}(C)}{T} + \Theta(1). \tag{26}$$

The final term is public-token replay per token. Recursion checks the native transcript, family updates, token absorption, and delegated commitment operations. Wider chunks reduce update frequency while increasing local SSD work.

For $n > 0$ evaluations of a size-$N_f = 2^{d_f}$ family, accumulation takes $O(nN_f)$ prover and $O((n+1)d_f)$ verifier field work, plus commitment/transcript operations. Storage is one polynomial (one scalar for a fixed source) and one evaluation claim. At fixed model, width, registry, and security parameters,

$$W_{\mathsf{prove}} = \Theta(K), \qquad W_{\mathsf{fin}} = \Theta(1), \qquad |\Pi| = \Theta(1), \qquad W_{\mathsf{verify}} = V_{\mathsf{crypto}}(C) + \Theta(T),$$

Here $V_{\mathsf{crypto}}(C)$ is terminal cryptographic verification cost, and $|\Pi|$ excludes the public script; cold setup is separate.

For full-chunk decode checkpoints $T_1 < \cdots < T_m$, let $W_{\mathsf{append}}(T_m, C)$ be the native and recursive work on the retained live branch. Producing and independently verifying all prefix proofs costs

$$W_{\mathsf{prefix}} = W_{\mathsf{append}}(T_m, C) + mW_{\mathsf{fin}}(C) + \Theta\left(\sum_{j=1}^{m} T_j\right),$$

$$V_{\mathsf{prefix}} = mV_{\mathsf{crypto}}(C) + \Theta\left(\sum_{j=1}^{m} T_j\right).$$

Each chunk is appended once; settlement and replay repeat at each checkpoint. The measurements use $m = 1$.

C.6 Optional Zero-Knowledge Wrapper

Before inference, authenticate a hiding, binding registration commitment $\rho_{\mathsf{reg}} = \mathsf{Com}_{\mathsf{reg}}(m; r)$ to the canonical encoding $m = (\mathsf{spec}, \mathsf{com}_\theta, \mathsf{com}_{\mathsf{tab}}, \mathsf{id}_{\mathrm{srs}})$, with secret r. Define

$$\begin{aligned} x_Z &= (\mathsf{spec}, \rho_{\mathsf{reg}}, \mathsf{scr}, g, \mathsf{id}_{\mathrm{srs}}), \qquad w_Z = (\mathsf{com}_\theta, \mathsf{com}_{\mathsf{tab}}, r, \Pi), \\ R_{\mathsf{wrap},T}(x_Z; w_Z) &= [\rho_{\mathsf{reg}} = \mathsf{Com}_{\mathsf{reg}}(m; r)] \\ &\quad \wedge [\mathsf{MProof.Verify}(x, \Pi) = 1]. \end{aligned} \tag{27}$$

The outer circuit checks the full predicate for Eq. (8), including canonical encodings and commitment-dependent setup. Its authenticated circuit keys are independent of hidden commitments. A complete, adaptive multi-theorem zero-knowledge argument of knowledge releases $\pi_Z \leftarrow \mathsf{Prove}_Z(pk_Z, x_Z; w_Z)$ with fresh randomness.

Extraction recovers the registration opening and core proof; binding fixes the registered commitments, and Theorem 5 gives execution integrity. Zero knowledge provides a simulator for the outer verifier's view,

$$\mathsf{View}_Z(x_Z; w_Z) \approx_c \mathsf{Sim}_Z(x_Z),$$

while registration hides the commitments. Tokens, length, program metadata, and SRS identity remain public. The circuit includes terminal cryptography and $\Omega(T)$ token replay; experiments measure the core protocol.

D Comparison Details

In Table 1, *API-only* restricts target access to queries, and *drift* denotes an output-distribution test. RUT uses a faithful local reference model (Zhu et al., 2026); instructional fingerprinting (IF) uses a publisher-enrolled instruction–response signature, specifically its black-box SFT variant (Xu et al., 2024).

Identity includes provenance identification by IF and authenticated model binding by Slalom or proofs. Slalom's remote-client construction authenticates the enclave and returns the output with a publicly verifiable model commitment (Tramèr & Boneh, 2019, Appendix B). Inference proofs bind computation to parameter commitments, which authenticated registration associates with the requested model (Sun et al., 2024; Qu et al., 2025; Zarinjouei et al., 2026; Wang et al., 2026). *Integrity* separately requires correctness of the individual execution.

For generation proofs, *E2E* spans token lookup, model execution, and prescribed output selection; *joint* checks multiple generated tokens together; *Inc.* extends an existing proof without reproving earlier model computation. zkLLM and zkGPT prove Transformer forward passes (Sun et al., 2024; Qu et al., 2025); zkRNN recursively composes recurrent transitions (Zarinjouei et al., 2026); zkAgent jointly checks completed transcripts through causal masking (Wang et al., 2026). MPROOF combines joint generation checks and incremental extension in bounded chunks. Dashes mark capabilities outside the cited task, including generation for Slalom's DNN evaluation.