Provable AI

A Claim-Bounded Criterion, a Semantic Boundary, and a Machine-Checked Construction

J. Konstapel — Constable Research, Leiden — September 2026 *Machine-checked artifacts produced in collaboration with Claude (Anthropic). All proofs in Appendix A were accepted by Lean 4.15.0 and can be re-checked by the reader in under a minute. Second edition, September 2026: revised after external review; every claim narrowed to its proved strength.*

Abstract

Current debate treats "provably correct AI" as either impossible in principle or as a property of symbolic systems that neural systems lack. This paper replaces both positions with a claim-bounded criterion. **Contribution 1** is an architectural reading of the modern transformer: in the open architectures examined here, four elements that displaced their alternatives under resource pressure form one coherent pattern, and two design elements of the pre-neural paradigm persist — the assigned token address and the continuous register. Independent measured work, cited but not claimed here, shows each is separately replaceable. **Contribution 2** is a provability criterion stated at its defensible strength: correctness is universally provable *relative to a formal specification* if and only if the claimed relation is formal and a sound derivation establishes it for every admissible input — architecture is irrelevant — together with a precise statement of the semantic boundary: Tarski excludes an internal total truth predicate for sufficiently expressive languages, and that obstruction, plus the instability of natural language itself, blocks any *total* certified meaning relation for open language, while partial and restricted relations remain formalizable. **Contribution 3** is a construction that lives inside the criterion: an answering system whose every claim is a derivation over a certified store, whose natural-language tier emits typed proposals rather than assertions, and which does not solve semantic undecidability but externalizes semantic settlement to an attributed human act whose provenance is formally preserved. **Contribution 4** is evidence in the only currency the argument permits: the system's core — balanced-ternary address kernel, truncation lemma, gate invariant, merge property, ledger step — is machine-checked in dependency-free Lean 4, resting on nothing beyond Lean's three standard axioms, with an explicit table separating what is proved from what is specified. The result is a class of AI that is non-fabricating, store-faithful and claim-bounded: every emitted content item is either backed by the certified store, with address, chain and certificates attached, or explicitly reported as vacant. It claims less than fluency, and proves what it claims.

1. Introduction and Claims

This paper makes four contributions and is explicit about the status of every statement in it.

New here: (i) an architectural reading of transformer convergence in the systems examined, and its diagnosis (Section 2); (ii) the claim-bounded provability criterion and the semantic boundary stated at its defensible strength (Sections 3–4); (iii) the claim-discipline construction with the attestation ledger (Sections 5–6); (iv) the accompanying machine-checked core (Section 7, Appendix A).

Cited, not claimed: ternary-weight parity at scale (Microsoft; independently Mila/Nolano and TII); parameter-free hash routing matching learned gates (Meta); vocabulary-free byte and character models (Google, Meta); the undefinability of truth (Tarski); proof-carrying code (Necula); verified systems at scale (seL4); the classical mathematics of balanced ternary and radix economy (Knuth).

Not claimed at all: that the construction runs at production scale (it does not yet); that its natural-language front end places questions well (unmeasured, and Section 4 shows why no total proof of it can exist for any system); that the content of the certified store is true of the world — the system certifies provenance and non-fabrication, and content itself only where a formal certificate covers it.

Terminology note. The paper occasionally uses the imagery of a net — strands, windings, closure — as orientation language from the author's prior work. No proof depends on it; every formal statement stands in standard terms.

2. The Convergence: Four Elements, Two Remainders

2.1 The reading

The evidence object is a single public file: the inference core of DeepSeek-V3 (`inference/model.py`, 808 lines). The same four elements are present in the other major open architectures examined (Llama, Qwen, Mistral); what follows is an architectural reading of these systems, not a survey claim about the entire field. In the systems examined, none of the four was adopted from a theory; each displaced an alternative that could not hold under pressure of energy, memory or gradient flow.

Position became rotation. `precompute_freqs_cis` encodes token position as phase on the complex unit circle (`torch.polar`), at a different frequency per dimension pair. The long-context correction functions literally compute in `num_rotations` per dimension. Absolute position encodings — the 2017 design — lost to this everywhere.

Dense computation became selection. The mixture-of-experts `Gate` scores 64 expert modules, keeps the top 6, and masks the rest to negative infinity. What does not pass contributes nothing. The balancing of this gate is itself drifting from learned score to deterministic count: DeepSeek-V3 replaced the auxiliary balancing loss by a per-expert bias updated by a counting rule, used only for selection.

Full caches became truncation hierarchies. Multi-head latent attention stores the past as a rank-512 latent and re-expands on read. The state lives compressed; the surface is derived. The engineering motive was memory; the structural result is a truncation rule.

Chains became one stream. Every layer computes `x = x + f(norm(x))` onto a single residual stream. Layers are not a causal chain; they are operations on one carrier, with normalization bounding each contribution.

The unified observation — these four are one pattern, selected by pressure rather than designed by theory — is offered as a diagnostic hypothesis: a reading of the systems examined, not an established result. To the author's knowledge it has no prior statement in the literature; a systematic search across the ternary, routing and tokenizer-free literatures found each element discussed separately and the convergence never named as a unit.

2.2 The two remainders, and their measured replaceability

In the architectures examined here, two design elements of the pre-neural paradigm persist. The **assigned address**: a vocabulary of 102,400 tokens whose meanings are free rows in a learned 102,400 × 2,048 table (~210 million parameters); in mT5-Base the vocabulary matrices are 256 million parameters, about 66 percent of the model. The **continuous register**: every weight a 16-bit float, every selection a softmax.

Both removals are already measured, by independent groups, and are cited as the empirical floor of this paper — not as its contribution. Byte- and character-level models remove the vocabulary: ByT5 cuts the vocabulary share from 66 percent to 0.1 percent; CANINE's deterministic hashed addresses beat the learned table by 2.8 F1 with ~28 percent fewer parameters; the Byte Latent Transformer matches Llama 3 at 8 billion parameters with no vocabulary at all. Hash Layers route with zero routing parameters at parity with learned gates (with the honest boundary that expert-choice routing beats hashing on quality). Ternary weights $\{-1, 0, +1\}$ reach parity with full precision at 2–4 billion parameters: BitNet b1.58 2B4T runs in 0.4 GB at 0.028 J per token against 2 GB and 0.258 J for its peer; Spectra replicates parity independently at 3.9 billion parameters; Falcon-Edge reproduces the recipe and releases the toolkit. One serious counter-measurement exists and is acknowledged: Ouyang et al. show the low-bit gap growing with training tokens for post-training quantization, with a small-scale native-ternary replication; the decisive side-by-side beyond 4 trillion tokens has not been run.

2.3 Why this section matters for what follows

This section proves nothing about provability, and the paper does not use it as if it did: Criterion P below is architecture-independent, so the criterion would hold on any substrate. The section's role is engineering motivation. If the field's own selection pressure is removing tables, scores and register depth, then a system whose identities are computed, whose selection is structural and whose register is minimal is the economically natural host for the construction of Sections 5–7 — the verified path becomes cheap exactly where the field is already heading. The two arguments are deliberately decoupled: convergence makes the construction attractive; the criterion, and only the criterion, makes it provable.

3. The Provability Criterion

Definition (formal task). Let Q be admissible inputs, Y admissible outputs, and R(q, y) a formally defined correctness relation. Let T be a sound proof system: everything T proves is true in the intended model.

Criterion P (claim-bounded provability). An AI system A has universally provable correctness *relative to a specification R* if and only if (a) R is a formal relation in a sound proof system, and (b) a derivation establishes R(q, A(q)) for every admissible q. The architecture of A is irrelevant to the criterion.

Justification. Direction (⇐) is the definition of formal provability. Direction (⇒): a universal formal proof of A's correctness is, by definition, a derivation establishing some formal relation between inputs and outputs over all admissible inputs; that relation is the R of (a). ∎

Stated this way, the theorem is close to a definition — deliberately so. Its content is not new mathematics but a boundary-drawing device: it makes "provable AI" a property of the *claimed*

relation, never of the architecture, and it exposes what fails when the claim is left unformalized. "This answer is true of the world", as ordinarily claimed for open language, names no formal R and so offers a proof system nothing to derive — which does not mean nothing about such a system is mathematical: a property can be partially formalized (a partial semantics, a model, an operationalization, an external specification), and Criterion P then applies to exactly the formalized part. Concretely: "this output is the exact string held at address a in this store" lies inside the boundary and is proved in Section 7; "this answer is true of the world" lies outside it, for any architecture. What the criterion adds over proof-carrying code and verified systems is not proof technique — those fields supply it — but the object: it names the claim interface as the design variable for AI systems. A specification counts as formal here exactly when (i) it is a relation written in the language of a sound proof system and (ii) its satisfaction on a given pair is checkable by that system's kernel. "This output is true because it comes from a reliable source" fails both tests — reliability names no relation and no checker — and is the standard false formalization: Case D wearing the vocabulary of Case A.

Four cases follow, and they cut across the symbolic/neural divide. **Case A** — task, semantics and evidence fully formal: universal correctness provable. **Case B** — an unreliable generator behind a sound verifier: correctness of accepted outputs provable, whatever the generator is, neural networks included. **Case C** — individual properties provable, unrestricted semantic correctness not: ordinary networks and transformers under open-language claims. **Case D** — the claim itself never formalized: nothing to prove.

The decisive distinction is therefore not symbolic versus neural. It is: formal specification plus sound verification, versus an unformalized semantic claim. A large language model is not intrinsically unprovable; its usual claim — "this answer is true of the world" — is unformalized. Restrict the claim and add a sound verifier, and provable properties appear. Section 5 carries this move to its limit.

A note on checker independence. Criterion P quantifies over sound proof systems; nothing in it privileges any one checker. Lean 4 is the instance used in Section 7 because it is small, fast to install and re-runnable by the reader; Coq, Isabelle or any other sound system would serve identically. The three axioms reported there (propositional extensionality, choice, quotient soundness) are the standard classical base of ordinary mathematics, not a special dependency of this construction. Trust in the checker itself is the one assumption this enterprise shares with the whole of formalized mathematics: Lean's kernel could in principle contain a defect, as could any checker's. The standard mitigations apply and are inherited, not invented here — a small kernel satisfying the de Bruijn criterion, per-theorem axiom reports (printed in Appendix B), and independent re-checking, feasible because the development is dependency-free and small enough to port to a second system. Soundness of this construction's own gate verifier (O4) is a separate, listed obligation.

4. The Semantic Boundary

Proposition B (the semantic boundary). (i) *(Tarski.)* No sufficiently expressive formal language contains a sound, total truth predicate for its own sentences. (ii) *(Obstruction.)* No design should rely on a *total*, certified meaning relation R_A(q, a) = "formal object a means the same as sentence q" over unrestricted natural language.

Argument. Part (i) is Tarski's theorem, cited, not reproved. Part (ii) is an obstruction argument, not a theorem, and the paper marks it as such. First, wherever a total meaning relation is required to determine truth conditions for a language expressive enough to describe its own semantics, it inherits the force of (i). Second, meaning and truth are not identical — a meaning relation may

concern reference, interpretation or use — and any *partial* or *restricted* formalization of meaning remains possible; Criterion P then covers exactly the formalized fragment, never the total relation. Third, natural language is not a fixed formal object: its expressions shift with time, place and speaker, so there is no stable total relation to axiomatize. The conclusion needed downstream is only this weaker, defensible one: no total certified R_A is available to build on, for any architecture — a symbolic coder, a search ranking and a transformer's weight matrix equally. A design must therefore keep R_A out of its claim set. (Part (i) relative to Tarski; part (ii) an obstruction, so marked.)

Three consequences fix the boundary's size and place.

First, the boundary is one predicate, not a fog. Everything an AI system does outside asserting R_A on open language is potentially inside mathematics. A design's quality can therefore be measured by how little of it stands on the wrong side.

Second, in a bare generative model under the unrestricted claim, effectively the whole system stands on the wrong side: generation samples from weights and no invariant separates supported from invented output. In the construction of Section 5, nothing stands on the wrong side, because the claim set never contains R_A.

Third, what cannot be proved can be either measured (empirical placement quality, with pre-registered thresholds) or **settled**: a human can decide, for one concrete question, which address answers it, and that decision — unlike the total relation — is a recordable, checkable fact. Section 6 builds on this.

Fourth, the boundary induces a ladder of claims, and everything below the top rung is inside mathematics: (1) total semantic correctness over open language — blocked by this proposition; (2) partial semantic guarantees relative to a formal object — consistency with a certified knowledge base, entailment within a fixed logic — provable exactly where formalized; (3) syntactic correctness — well-formedness of code or structured output against a grammar — provable; (4) provenance — this output is what the store holds at this address — provable, and the rung this construction occupies end-to-end. A system's honest place is the highest rung it can certify, not the rung its description names. RAG, restricted to retrieval claims, stands on rung 4 for the retrieved part; its generation stands on none (Section 8).

5. The Construction: Claim Discipline

Design principle. A provable AI is not an AI whose every behaviour is proved — Proposition B blocks that machine. It is an AI whose every **claim** is proved. That machine exists, and this section specifies it.

The one claim. Every output asserts exactly: *this output is the content held by the certified store at the computed address, reached by the recorded derivation, under the certificates attached.* Formally: Output(q) = y implies Derive(S, q, y). The claim never includes "y is true of the world" and never includes R_A. By Criterion P, a system whose entire claimed behaviour is Derive is provable in full — there is no residue, because nothing outside the relation is claimed.

Store-membership, not world-truth. Derive asserts that the output is exactly what the certified store holds at the computed address. It does not assert that the store's content is true of the world: a certified store can hold a wrong answer, and the machine-checked gate theorem of Section 7 proves store-membership, nothing more. On the proof-carrying tier the content itself carries a machine-

checked certificate, so correctness relative to the checker is certified; on receipted tiers the guarantee is provenance. The accurate vocabulary, used from here on, is therefore: the system is **non-fabricating**, **store-faithful** and **claim-bounded**. The word "correct" is reserved for content covered by a formal certificate.

Claim versus behaviour. The claim is carried by the output type: no constructor exists without an address, so every emission asserts Derive by its form (Section 7). Behaviour is the function that produces the emission, and the gate theorem proves the reference function satisfies the claim its type asserts. A production bug is therefore not a false claim but a divergence from the proved reference — exactly the gap the production-path invariant names, undischarged in the status table, and the reason differential testing against the reference is obligation (b) of the work programme.

Components. (C1) A kernel of computed identities: balanced-ternary integer addresses, with unique representation, truncation-as-rounding, and union-merge — proved in Section 7. (C2) A certified store: triples (address, content, certificate); content enters only with a receipt, formal content only with a checked proof certificate. (C3) A loop as the only output path: read the store at the address and at its structural images; emit content tagged with address and chain, or the token VACANCY; no operation fabricates content. (C4) A minimal verifier — the trusted computing base, deliberately small so it can itself be verified — through which all admission and emission pass; every producer outside it may be arbitrarily unreliable without breaking the claim. This is proof-carrying code, applied to answers. (C5) An append-only, hash-chained attestation ledger (Section 6).

Two tiers of input. Formal inputs (addresses, structured queries, machine-checkable proofs, receipted content) have a defined right address — computed by the kernel rule — and the coder emits it with its computation trace as certificate; end-to-end correctness is then a derivation, coder included. Open natural language has no certifiable address (Proposition B); the front end therefore emits output typed as PROPOSAL, presenting the address, its content and its neighbours. The user, not the machine, closes the semantic gap — standing before an open, navigable shelf rather than an oracle. The machine's claim about the proposal remains Derive, and remains proved.

6. The Attestation Ledger: Carrying the Unprovable

Rule. When a user confirms that address a answers question q, the system appends (canon(q), a, who, when) to the ledger, hash-chained to its predecessor; canon is a fixed, deterministic, non-semantic normalization. From then on, the same canonical question carries a certificate — not "a means q", which nothing can certify, but "a was settled for canon(q) by *who* on *when*", a checkable formal fact.

What the ledger does, and does not do. It does not solve semantic undecidability, and it does not prove that the settling human interpreted q correctly — nothing can prove that. It externalizes semantic settlement to an attributed human act and formally preserves the resulting provenance: the settlement becomes an auditable, attributable, immutable *event*, never a proved truth. Signatures and cryptographic chain integrity belong to the production specification; the machine-checked core models the entry and the chain step only, as the status table in Section 7 states.

Canon and governance, stated as requirements. The normalization canon is itself part of the trusted base of settlement: it must be deterministic, published and versioned, and every ledger entry records the canon version under which it was made — so two communities normalizing differently merge as attributed data instead of fragmenting silently. On top of the ledger, a minimal default policy suffices for downstream use and is strictly optional: a community designates who may settle, and treats as operative the most recent entry by a designated settler, with all other entries remaining

visible as data. Any richer policy (quorum, roles, recency windows) replaces this default without touching the claim set.

Three exact consequences.

1. **The burden is recorded, not discharged.** The semantic judgment happens once, by a named human, outside the claim set, and leaves a formal trace. What is certified is the event of settlement — never the correctness of the interpretation. Every claim the system itself makes remains provable.
2. **The unprovable tier drains monotonically into the proved tier.** Every settled question is thereafter a formal input: ledger lookup, certificate attached. A community using the system builds, as a by-product, a certified question–answer map of its own domain, mergeable with any sibling by set union. Monotonicity here is by construction — entries only accumulate; the *rate* of growth is an empirical matter, depends on human activity, and is unmeasured.
3. **Disagreement is data.** Two users settling the same question on different addresses yield two attributed entries, localized to one canonical question. The system holds both and says so; it never adjudicates meaning.

To the author's knowledge, this mechanism — recording instance-by-instance human settlement of the in-principle-unformalizable meaning relation as attributed, chained, checkable events, with monotone growth of the certified tier — has no prior publication. Its nearest relatives are discussed in Section 8.

Settlement at scale. Three observations bound the human cost. Question traffic is heavily skewed: a small head of frequently asked canonical questions carries most volume, and each of these needs settling once, ever — after which it is a certified lookup for every later asker, in every merged sibling. Settlement can be machine-assisted without weakening the claim: a generator may draft proposals and even suggest settlements, but only a signed human entry — or an entry explicitly typed as machine-attested — enters the ledger, so the record never overstates its own authority. And conflicting settlements need no adjudication mechanism, because the ledger is not a consensus system: it is a register of attributed judgments. Any community may layer its own resolution policy (quorum, roles, recency) on top of the ledger without touching the claim set, and two communities with different policies still merge by union.

7. The Machine-Checked Core

The construction's kernel and loop invariants are not argued; they are checked. Appendix A contains the complete Lean 4 development — dependency-free: no mathlib, no external libraries — and Appendix B the check harness. Lean 4.15.0 accepts every theorem; the axiom report shows, per theorem, only Lean's three standard axioms (propositional extensionality, choice, quotient soundness) and no `sorry`.

Checked, against running definitions:

- **decode_encode** — every integer has the encoding the kernel computes, and decoding returns it: the computed identity is a bijection (Theorem 1 of the specification).
- **tail_bound** — the value of any k-trit tail lies two-sidedly within ±(3^k − 1)/2 of zero at the kept scale: truncation is rounding to nearest, so refinement of depth never invalidates an issued address (Theorem 2).
- **merge_is_union** — membership in a merged store is membership in either part: federation without translation (Theorem 3).

- **answer_store_backed** and **vacancy_means_empty** — the gate invariant in both directions: emitted content exists verbatim in the store at its address, and a vacancy is emitted only when the address is truly empty. Fabrication and suppression are both impossible — as properties of the function, not intentions of the builder (Theorem 4).
- **chain_deterministic** — one ledger step is a congruence: equal history and equal entry give equal successor. This is *ledger-step determinism*, and it is mathematically slight by design; it is not ledger integrity. It establishes neither collision resistance, nor tamper evidence, nor append-only behaviour, nor author authenticity, nor any whole-chain property — those are O5 and the cryptographic layer, both undischarged (see the status table below).
- **Output type** — no output constructor exists without an address: claim completeness by construction.

Executable self-checks close the loop on the paper's own numbers: `encode 43 = [+,−,−,−,+]` (the worked example of the specification), and the round trip of −144719454, the content-derived address of the author's prior provenance essay, through the proved kernel.

Scope, stated plainly. These are proofs about the reference core in Appendix A — a real, running kernel and loop, but small: the store is a list, the loop implements direct lookup. The production system's kernel must be differentially tested against this proved reference; the full-chain ledger induction (O5), the verifier-soundness proof (O4) and the production-path invariant are specified, bounded obligations, not yet discharged. The paper claims exactly what is checked, and lists what is not.

Status table — specified versus machine-checked.

Property	Specified	Machine-checked (Appendix A)
Computed address, bijection (O1)	yes	yes
Truncation as rounding (O2)	yes	yes
Output store-backed; no fabrication	yes	yes
Vacancy only when address truly empty	yes	yes
Merge = union (T3)	yes	yes
No output without an address (claim completeness)	yes	yes
Ledger-step determinism	yes	yes — mathematically slight
Whole-chain ledger induction (O5)	yes	no
Cryptographic chain integrity, signatures	yes	no
Verifier soundness (O4)	yes	no
Production-path invariant	yes	no
Semantic correctness of stored content	deliberately not claimed	no

8. Relation to Prior Work

Retrieval-augmented generation (RAG). RAG conditions a generator on retrieved documents and often cites them. The resemblance is superficial and the difference is the point: in RAG the generator remains the output path, so no invariant separates retrieved fact from generated content —

hallucination with citations remains possible and is documented. The nearest empirical neighbour is the attributed and verifiable generation line inside RAG: citation generation and post-verification at passage, sentence and sub-sentence grain (attributed question answering, Bohnet et al. 2022; citation-quality evaluation, Gao et al. 2023; verification-driven pipelines such as LLatrieval 2023 and VeriCite 2025; fine-grained reference–claim interleaving reporting around ninety percent citation accuracy, 2024). These works pursue the same goal — checkable answers — by statistical means, with the generator still on the output path; the field has begun to state the residual gap in its own words: citation correctness is not faithfulness (ICTIR 2025). The construction here replaces the statistical guarantee by an invariant. Here generation never reaches the output; the loop of Section 5 is the only path, and its invariant is machine-checked. RAG is Case C of Criterion P; this construction is Case A on its claim set, with Case B available when a generator is placed in front as proposer. This placement is relative to the claim set — deliberately so, since the criterion's entire content is that provability attaches to claims: nothing prevents a RAG stack from adopting this claim set and gate. The contribution is the disciplined claim set with its machine-checked invariant, not a property forever unavailable to others.

Proof-carrying code and verified systems. Necula's pattern — untrusted producers, small verified checker — and seL4's demonstration that a real system can be verified end-to-end are the engineering ancestry of C4. The transfer here is from code safety to answer provenance, plus the attestation mechanism for the semantically open part, which PCC does not address.

Formal-verification-of-ML. Existing work proves properties of networks (robustness bounds, fairness constraints): Case C. Criterion P explains both its value and its ceiling, and why this paper verifies the claim set rather than the network.

Tokenizer-free, deterministic-routing and ternary literatures. These supply Section 2's measured floor and are credited throughout; none states the convergence as a unit, and none connects it to provability.

Positioning table.

System class	Formal claim about output	Machine-checked today	Where the semantic boundary sits	Trusted base
Generative LLM,	none ("true of the world" is	properties of the network at most (Case	inside the generator; no invariant marks it	the whole model
RAG with citations	none universal; citation is	typically nothing on the output path	inside the generator; retrieved and invented	the whole model plus
Proof-carrying code	accepted code satisfies the	verifier and proof checking	out of scope (code, not open language)	small verifier
seL4-class verified	implementation meets its	end-to-end	out of scope	proof toolchain and spec
Knowledge base with	curated provenance,	typically none	in the curators, not recorded per claim as a	platform and process
This construction	Output(q) = y ⇒ Derive(S, q,	kernel, gate, merge, ledger step (Appendix A); Q4/Q5 specified	outside the claim set; settled per instance, recorded in the ledger	minimal verifier (C4) plus proof

9. Limitations and Open Measurements

Stated once, without qualification elsewhere. (1) The construction runs as a proved reference core plus a specification; the production system it targets (a 3.6-million-item address store with a public

interface) has not yet been placed behind the proved loop. (2) Proposal quality on open language is unmeasured; a pre-registered twenty-item placement test with a fixed threshold is specified for it, and Proposition B entails this number governs comfort, not correctness. The per-question human settlement cost is bounded by the skew and machine-assistance arguments of Section 6, but has not yet been observed on a live population. (3) Obligations O4 (verifier soundness) and O5 (full-chain ledger induction) are specified but undischarged. (4) The ternary register is fixed for economy on cited evidence; provability is register-independent, and the token-axis question raised by Ouyang et al. remains open in the field. (5) Criterion P is definitional in character — a boundary-drawing device, not new mathematics — and part (ii) of Proposition B is an obstruction argument, not a theorem; both are stated as such where they appear.

Work programme, in order. (a) Discharge O4, O5 and the production-path invariant; replace the list store in the reference core by a verified finite map, so the proofs approach the target scale; caching and concurrent settlement follow the same rule — admitted only behind a proved invariant. (b) Differentially test the production kernel against the proved reference on exhaustive small ranges and random large ranges, wired into a continuously run check the reader can execute, with property-based tests beside the theorems. (c) Run the pre-registered twenty-item placement test, and measure or simulate the question-skew assumption on a realistic corpus, since the settlement-cost argument of Section 6 depends on it. (d) Publish canon as a versioned open specification. (e) Keep governance an optional layer, so the claim set stays fixed while communities vary.

10. Conclusion

The transformer's history, in the architectures examined here, is a sequence of removals, and its two persistent pre-neural elements — the assigned address and the continuous register — are already separately removed in measured work. On that substrate this paper contributes: a claim-bounded criterion showing that AI provability is a property of claimed relations, not architectures; a statement of the semantic boundary at its defensible strength; a construction that claims only what it derives and records human settlement of the semantically open part as attributed, chained, checkable events; and a machine-checked core the reader can re-verify in a minute, with an explicit table separating the proved from the specified. Conventional generative systems provide no universal machine-checkable guarantee for the unrestricted truth of their natural-language outputs — under that claim there is nothing to derive. The construction here guarantees something narrower and proves it: every emitted content item is either backed by the certified store, with address, chain and certificates attached, or explicitly reported as vacant. It says less, and proves more. Both classes will exist. The contribution of this paper is that the second class now has its criterion, its design, and its first checked artifact — with its claim layer cut to the exact size of its proved layer.

Annotated References

1. **Tarski, A., "The Concept of Truth in Formalized Languages" (1936).** *Why read?* The undefinability theorem behind part (i) of Proposition B; part (ii) of Section 4 extends it only as an obstruction argument, and is marked as such. *Advice:* a modern exposition suffices.

2. **Necula, G., "Proof-Carrying Code" (POPL 1997).** *Why read?* The verifier pattern of component C4: untrusted producers, small checked verifier. This paper transfers it from code to answers. *Advice:* introduction only.

3. **Klein, G. et al., "seL4: Formal Verification of an OS Kernel" (SOSP 2009).** *Why read?* Existence proof that end-to-end machine verification of a real system is feasible; the obligation list here is far smaller. *Advice:* abstract and lessons learned.

4. **De Moura, L. & Ullrich, S., "The Lean 4 Criterion Prover and Programming Language" (CADE 2021).** *Why read?* The checker of Appendix A; the reader's one-minute re-verification runs on it. *Advice:* skim for scope.

5. **Knuth, D., "The Art of Computer Programming", Vol. 2, §4.1.** *Why read?* Balanced ternary — unique representation, truncation-as-rounding — and the radix-economy result; the classical floor under the kernel. *Advice:* three pages, self-contained.

6. **DeepSeek-AI, "DeepSeek-V3 Technical Report" (arXiv 2412.19437) and the public `inference/model.py`.** *Why read?* The evidence object of Section 2; every element of the reading can be checked against a line of the file. *Advice:* read `precompute_freqs_cis`, `Gate.forward`, `MLA.__init__`, `Block.forward`.

7. **Wang, L. et al., "Auxiliary-Loss-Free Load Balancing for Mixture-of-Experts" (arXiv 2408.15664).** *Why read?* The learned score displaced by a counting rule, in production — the drift Section 2 diagnoses. *Advice:* short.

8. **Xue, L. et al., "ByT5: Towards a Token-Free Future" (TACL 2022); Clark, J. et al., "CANINE" (TACL 2022); Pagnoni, A. et al., "Byte Latent Transformer" (ACL 2025).** *Why read?* The measured removal of the assigned address, at three scales. *Advice:* ByT5 Table 1 first.

9. **Roller, S. et al., "Hash Layers for Large Sparse Models" (NeurIPS 2021); Zhou, Y. et al., "Expert Choice Routing" (NeurIPS 2022).** *Why read?* Deterministic routing at parity, and its honest quality boundary. *Advice:* read together.

10. **Ma, S. et al., "The Era of 1-bit LLMs" (arXiv 2402.17764) and "BitNet b1.58 2B4T Technical Report" (arXiv 2504.12285); Kaushal, A., Vaidhya, T. et al., "Spectra" (ICLR 2025); TII, "Falcon-Edge" (2025).** *Why read?* Ternary parity, its energy bill (0.028 vs 0.258 J per token), and two independent replications. *Advice:* tables only.

11. **Ouyang, X. et al., "Low-Bit Quantization Favors Undertrained LLMs" (ACL 2025).** *Why read?* The one serious counter-measurement to register minimalism; this paper cites it rather than argues around it. *Advice:* note the PTQ/native distinction.

12. **Lewis, P. et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP" (NeurIPS 2020).** *Why read?* The nearest-looking prior system class; Section 8 states exactly why the resemblance is superficial. *Advice:* architecture figure suffices.

13. **Konstapel, J., "The Address of a Proof" (Constable Research, 2026).** *Why read?* The running admission path — content-derived address, receipt, Lean certificate — that instantiates component C2; the round-tripped number of Section 7 is this essay's own address. *Advice:* postscript for the self-test.

14. **Konstapel, J., "Where Computation Comes From" and "The Six Theorems of the MAZE, Second Edition" (Constable Research, 2026).** *Why read?* The extended versions of Sections 2 and 5–6, with full derivations and figures. *Advice:* companion reading; this paper is self-contained without them.

15. **The attributed/verifiable-generation line inside RAG — Bohnet et al., "Attributed Question Answering" (2022); Gao et al., "Enabling Large Language Models to Generate Text with Citations" (2023); "LLatrieval: LLM-Verified Retrieval for Verifiable Generation" (2023); "Ground Every Sentence" (2024); "VeriCite" (SIGIR-AP 2025); "Correctness is not Faithfulness in Retrieval Augmented Generation Attributions" (ICTIR 2025).** *Why read?* The nearest empirical neighbour of this construction: the same goal, checkable answers, pursued statistically — citations generated and post-verified to roughly ninety percent accuracy at sentence grain — with the generator still on the output path. The last title is the field naming the residual gap itself. *Advice:* read one beside Section 8 to see exactly where the empirical line stops and the invariant begins.

Appendix A — The Machine-Checked Core (MazeAI.lean, verbatim)

Accepted by Lean 4.15.0, dependency-free. Axiom report per theorem: `propext`, `Classical.choice`, `Quot.sound`; no `sorryAx`.

```
/-
  maze.ai/1 — Machine-checked core (Lean 4.15.0, no external
libraries)
  J. Konstapel & Claude, Constable Research, Leiden —
September 2026

  Discharged in this file, against running definitions:
    O1  kernel bijection      : decode (encode n) = n
(Theorem 1)
    O2  truncation lemma      : 2·|tail| ≤ 3^k − 1
(Theorem 2)
    T3  merge is union        : membership in append
(Theorem 3)
    O3  loop invariant        : every emission is store-
backed   (Theorem 4)
    O5' ledger step           : chain hash determined by
entry
    O6  claim completeness    : every output carries its
address (by type)
-/

namespace MazeAI

/-- One winding against its reference: below, on, above. -/
inductive Trit where
  | neg | zero | pos
deriving Repr, DecidableEq
```

```
/-- The integer one trit denotes. -/
def Trit.val : Trit → Int
  | .neg => -1
  | .zero => 0
  | .pos => 1

/-- Little-endian balanced-ternary decoding: shallow trit
first. -/
def decode : List Trit → Int
  | [] => 0
  | t :: ts => t.val + 3 * decode ts

/-- Balanced-ternary encoding of an integer. -/
def encode (n : Int) : List Trit :=
  if n = 0 then []
  else if n % 3 = 0 then .zero :: encode (n / 3)
  else if n % 3 = 1 then .pos :: encode ((n - 1) / 3)
  else .neg :: encode ((n + 1) / 3)
termination_by n.natAbs
decreasing_by all_goals omega

/-- **O1 — Theorem 1 (kernel bijection).** Decoding an
encoding returns the
    number itself, for every integer. -/
theorem decode_encode (n : Int) : decode (encode n) = n := by
  induction n using encode.induct
  all_goals unfold encode
  all_goals simp_all [decode, Trit.val]
  all_goals omega

/-- Powers of three, defined from first principles. -/
def pow3 : Nat → Int
  | 0 => 1
  | k + 1 => 3 * pow3 k

/-- **O2 — Theorem 2 (truncation is rounding).** The value of
any k-trit tail
    lies strictly within half a unit of the kept scale: 2·|
value| ≤ 3^k − 1,
    stated two-sidedly over the integers. -/
theorem tail_bound (ts : List Trit) :
    2 * decode ts ≤ pow3 ts.length - 1 ∧
    -(pow3 ts.length - 1) ≤ 2 * decode ts := by
  induction ts with
  | nil => simp [decode, pow3]
```

```
  | cons t ts ih =>
      obtain ⟨ih1, ih2⟩ := ih
      have hv : t.val = -1 ∨ t.val = 0 ∨ t.val = 1 := by
        cases t <;> simp [Trit.val]
      rcases hv with h | h | h <;>
        simp only [decode, List.length_cons, pow3, h] <;>
omega

/-- A store: pairs of address and content. -/
abbrev Store := List (Int × String)

/-- **O6 — the output type.** Every constructor carries its
address; no other
    output can be built. Claim completeness holds by
construction of the type. -/
inductive Output where
  | derived (addr : Int) (content : String)
  | vacancy (addr : Int)
deriving Repr, DecidableEq

/-- The answer loop: the only output path. It reads the
store; it fabricates nothing. -/
def answer (s : Store) (addr : Int) : Output :=
  match s.find? (fun p => p.1 == addr) with
  | some p => .derived p.1 p.2
  | none => .vacancy addr

/-- **O3 — Theorem 4 (the gate cannot assert what it does not
hold).**
    If the loop emits content, that exact (address, content)
pair is in the store. -/
theorem answer_store_backed (s : Store) (a addr : Int) (c :
String)
    (h : answer s addr = .derived a c) : (a, c) ∈ s := by
  unfold answer at h
  cases hf : s.find? (fun p => p.1 == addr) with
  | none => simp [hf] at h
  | some p =>
      simp [hf] at h
      have hp := List.mem_of_find?_eq_some hf
      obtain ⟨h1, h2⟩ := h
      subst h1; subst h2
      simpa using hp
```

```
/-- **O3, second half.** A vacancy is emitted only when no
entry holds the address. -/
theorem vacancy_means_empty (s : Store) (addr a : Int)
    (h : answer s addr = .vacancy a) :
    a = addr ∧ ∀ c, (addr, c) ∈ s → False := by
  unfold answer at h
  cases hf : s.find? (fun p => p.1 == addr) with
  | some p => simp [hf] at h
  | none =>
      simp [hf] at h
      refine ⟨h.symm, ?_⟩
      intro c hc
      have hn := List.find?_eq_none.mp hf (addr, c) hc
      simp at hn

/-- **Theorem 3 (merge is union), over stores.** -/
theorem merge_is_union (sA sB : Store) (p : Int × String) :
    p ∈ sA ++ sB ↔ p ∈ sA ∨ p ∈ sB := List.mem_append

/-- A ledger entry: canonical question, settled address, who,
when. -/
structure Entry where
  canonQ : String
  addr : Int
  who : String
  date : String
deriving Repr, DecidableEq, Hashable

/-- Chain step: the running hash commits to the whole entry
and the past. -/
def chainStep (prev : UInt64) (e : Entry) : UInt64 :=
  mixHash prev (hash e)

/-- **O5 (step form) — ledger integrity.** The chain value is
a function of
    the past and the exact entry; equal links with equal
pasts force equal steps. -/
theorem chain_deterministic (prev : UInt64) (e e' : Entry)
    (hq : e = e') : chainStep prev e = chainStep prev e' :=
by
  subst hq; rfl

end MazeAI
```

Appendix B — Check Harness (check.lean) and Expected Output

```
import MazeAI
#print axioms MazeAI.decode_encode
#print axioms MazeAI.tail_bound
#print axioms MazeAI.answer_store_backed
#print axioms MazeAI.vacancy_means_empty
#print axioms MazeAI.merge_is_union
#print axioms MazeAI.chain_deterministic
-- Executable sanity: 43 = + - - - + (little-endian: + - - -
+), decode round-trip
#eval MazeAI.encode 43
#eval MazeAI.decode (MazeAI.encode 43)
#eval MazeAI.decode (MazeAI.encode (-144719454))
```

Reproduction (Lean 4.15.0, bare toolchain from github.com/leanprover/lean4 releases):

```
lean MazeAI.lean                          # exit 0 = all proofs
accepted
lean -o build/MazeAI.olean MazeAI.lean
LEAN_PATH=build lean check.lean
```

Expected check output: each `#print axioms` line lists only `propext`, `Classical.choice`, `Quot.sound` (subsets thereof); the evaluations print `[MazeAI.Trit.pos, MazeAI.Trit.neg, MazeAI.Trit.neg, MazeAI.Trit.neg, MazeAI.Trit.pos]`, `43`, and `-144719454`.