TensorCash Inference Verification: Mathematical Specification (v1)

Version 1.0 — mainnet-inception specification; open research v1.

ABSTRACT

TensorCash secures a blockchain by tying its proof-of-work directly to inference of a publicly identified, open-source large language model. Each block carries a *proof object* — a record of the model's sampling trajectory on a hash-derived prompt — that any other node must be able to **verify** before accepting the block. Verification cannot simply re-execute the miner's forward pass and compare bit-for-bit: at the floating-point precisions used in production inference (fp16, bf16, fp8), the same model run on different hardware, with different batch sizes, or under different attention kernels does not produce identical logits. A useful verifier must therefore be a **statistical decision rule** whose null hypothesis is "the miner ran the declared model honestly", whose alternative space is "the miner did anything else", and whose Type-I error is bounded by explicit calibrated gates.

This document specifies the decision rule and model-admission process that TensorCash mainnet is intended to ship with at inception. The two surfaces are deliberately separate. **Model admission** decides whether a model identifier and difficulty scalar enter the registry. **Proof verification** decides whether a particular block's proof object is accepted. Per-block verification proceeds along a three-tier ladder — Quick, Smell, and Full — each tier strictly tightening the set of accepted proofs while gating a different network-level action: header propagation, header gossiping, and block acceptance respectively.

The protocol described here is intentionally an **open research v1**. Verification of probabilistic compute is an active area of research and the gates we specify — calibrated null distributions, fraction-based acceptance regions, fingerprint Mahalanobis tests, snapped 74-dimensional bootstraps — are the best instruments we have today, not a finished science. The network is designed to admit more sophisticated verification rules for future blocks: tighter bootstraps, closed-form nulls for snapped Gaussians, model-specific correlation matrices, gates that exploit attestation rather than statistical inference, and so on. Backward compatibility in this document means *"v1 will not become impossible to support"*, not "v1's individual verdicts are permanent": improved verifiers are explicitly invited for future verifier versions.

TABLE OF CONTENTS

I. Setting and Notation

We fix throughout a causal language model f_θ with vocabulary size V, declared compute precision dt ∈ {fp16, bf16, fp8, fp32}, and a content-addressed identifier (repository name and commit hash). At each step t of a sampling trajectory of length W, the model emits a logit vector $L_t \in \mathbb{R}^V$, from which a single token τ_t is drawn under hyperparameters (τ, p, k, ρ) — temperature, top-p, top-k and repetition penalty.

A miner submits a *proof object* P containing, for every step: the chosen token τ_t and its probability p_t under the truncated sampling distribution; a deterministic uniform draw $u_t \in [0, 1)$; the top-K logits $\tilde{L}_t \in \mathbb{R}^K$ together with their indices $S_t \in \mathbb{N}^K$ (we take $K = 70$); the means of the sorted full logit vector over four rank buckets *B* = {[0, 50), [50, 500), [500, 2000), [2000, *V*)} and the global mean. The proof also carries the cryptographic scaffolding of the block: a verifiable delay function output (VDF), the tick at which the VDF was computed, the prefix of the block header, the previous block hash, and the final hash that is compared against the difficulty target. In the mainnet-inception FlatBuffers schema the bucket/global summaries are carried as `logsumexp_stats`; the same proof object also carries the prompt tokens, pad mask, softmax normalisers, declared target, IPFS CID, and auxiliary flags needed by the verifier and registry tooling.

This is a deliberate portability boundary. The proof is expressed in terms of externally meaningful transcript data — token IDs, sampler draws, top-logit summaries, rank-bucket summaries, model identifier, precision tag, and cryptographic chain state — rather than intermediate activations, attention maps, hidden states, or layer-specific traces. That keeps the verifier portable across transformer variants and future autoregressive model families. Architecture-specific internal evidence may be useful for audits, but it is not part of the v1 block-proof interface.

We write Φ for the standard normal CDF, χ^2_d for the chi-squared with d degrees of freedom, and snap(*x*; *u*) = *u* · round(*x*/*u*) for the projection of x onto the lattice $u\mathbb{Z}$. The half-unit-in-last-place at value x in dtype dt is denoted $\mathrm{ulp}(x; \mathrm{dt})$ and equals $2^{\lfloor \log_2 |x| \rfloor - m - 1}$ for m mantissa bits.

II. The Sampling Contract

Before describing the verifier, we record the constraints on the hyperparameters under which a proof is *eligible* for verification at all. Two of these constraints — the bounds on (τ, p, k, ρ) and the exclusion of low-entropy generations — are themselves part of consensus.

i. Proof-window length and proof-size budget

At mainnet inception the solution window is $W = 256$ generated tokens. This is a protocol compromise between statistical power, miner ergonomics, verifier cost, and network relay size. Shorter windows reduce the evidence available to the Smell and Full gates and weaken the collision-resistance intuition. Longer windows improve statistical power but expand the proof object, increase block relay burden, and make retries more expensive for honest miners.

The proof is intentionally a bounded transcript rather than an unlimited conversation log. With prompt/context tokens, chosen tokens, deterministic sampler values, top-logit summaries, bucket statistics, normalisers, model metadata, and cryptographic scaffolding, practical proof objects sit in the rough range of 160 KB to 1 MB depending on schema choices, compression, and implementation details. The limited output length is therefore not arbitrary: it is the sweet spot where the proof remains large enough to be statistically useful while still small enough to carry through the block and gossip layers.

ii. Bounded sampler hyperparameters

Each proof must satisfy

$$5 \le k \le 50, \qquad 0.1 < p \le 1, \qquad 0.25 < \tau < 2, \qquad 0.1 < \rho \le 1.$$

A proof outside any of these ranges is rejected by the Quick stage without further analysis.

Rationale. The lower bound on k guarantees that the truncated support has enough cardinality for the smell-test fingerprint to be informative — at $k < 5$ the top-k distribution collapses to a degenerate trio and the inert-token presence test in the Smell stage becomes vacuous. The upper bound on k matches the persisted top-K width: requiring $k \le 50 < 70 = K$ ensures that the truncated sampling distribution is fully reconstructable from the persisted logits without further model evaluation. The temperature window $0.25 < \tau < 2$ excludes both near-greedy decoding (which contains essentially no statistical information beyond an argmax check, defeating the bootstrapped null in Full-stage statistical test) and near-uniform decoding (which inflates the test variance to the point where the verifier loses discriminative power against tampered proofs). The bounds on top-p and on the repetition penalty are chosen analogously.

iii. Low-entropy exclusion

Let $\bar{p} = \frac{1}{W}\sum_t p_t$ be the mean of the proof's chosen probabilities. We require

$$\bar{p} < 0.925.$$

Rationale. The proof's chosen probability p_t is bounded below by $1/k$ and above by 1. A proof whose mean chosen probability is close to 1 describes a sequence in which the model was almost certain at every step. In that regime the Quick-stage sampling test reduces to checking that the most probable token was selected — a statement that any plausible LM, including a small distillation or even a heuristic next-token-frequency table, will satisfy on common short prompts. Worse, the per-step Mahalanobis test in the Full stage loses statistical power as the top logit pulls away from the rest: the discrete component of the 74-dimensional observation is dominated by a single coordinate, the covariance becomes effectively rank-1, and the bootstrap null collapses onto its mode. Forbidding $\bar{p} \ge 0.925$ ensures that every accepted proof exercises a non-trivial distributional regime in which the verifier's decision rule has measurable power.

The choice of 0.925 is conservative: it rejects only the most degenerate proofs while leaving room for short, high-confidence segments inside otherwise diverse trajectories. A future version of the verifier may replace this scalar threshold with a per-step entropy lower bound, preserving backward compatibility.

III. Model Admission at Mainnet Inception

Model admission is the registry-side process that decides whether a candidate model identifier, commit, content address, and difficulty scalar may be used for future proof-of-inference blocks. It is distinct from proof verification: admitting a model does not accept any block, and rejecting a proof does not by itself remove the model from the registry.

At mainnet inception, admission has four components: an advisory model audit, an on-chain deposit and commit threshold, off-chain public discussion threads, and an on-chain challenge path for registered models. The network is open to more sophisticated admission and verification rules in future versions, but this section describes the mechanism that ships first.

i. Advisory model audit

The model audit calibrates a single chain-level scalar — the *difficulty* parameter D — to the actual computational cost of the declared model. Because difficulty is a chain-level constant rather than a per-block quantity, the audit runs at registration time, not on every block.

ii. Difficulty as an inverse compute scalar

The base proof-of-work target T_{base} is adjusted on a per-model basis through

$$T_{\text{adj}} \le T_{\text{base}} \cdot \frac{N}{D},$$

where N is a global normalizer ($N = 10^6$ in v1) and D is the difficulty scalar registered for the model. Since smaller targets correspond to harder hash puzzles, *higher* D corresponds to a *harder* hash PoW. A model that requires more compute per token must therefore be paired with a *lower* D to compensate.

Calling the model's measured FLOPs per generated token F, and choosing a fixed reference model with F_{genesis}, the audit-prescribed difficulty is

$$D^\star = N \cdot \frac{F_{\text{genesis}}}{\widehat{F}}.$$

The chain-level safety bound is one-sided. Because consensus enforces an adjusted target proportional to N/D, a claimed difficulty below the audit prescription makes the hash puzzle easier than the measured compute cost warrants, while a claimed difficulty above the prescription only self-penalises miners of that model. The admission rule therefore treats the lower bound as consensus-critical:

$$D_{\text{claim}} \ge (1 - \varepsilon)\, D^\star,$$

with default tolerance $\varepsilon = 0.05$. Operator audit tooling may still flag large positive deviations from $D^\star$ as economic or liveness warnings, but the model-safety failure is an under-claim that would enlarge the adjusted block target.

iii. Context-length fairness and amortisation

The model-difficulty scalar is deliberately approximate. It prices the declared model by measured effective FLOPs per generated token; it does not attempt to discriminate by prompt length, user context length, retrieval payload size, or the amount of prefill work already performed before a 256-token proof window begins.

That omission is intentional. Context length is miner- and user-chosen, privacy-sensitive, highly variable, and easy to game if made a per-block fairness input. In ordinary autoregressive serving, a miner can pay an initial prefill cost for a long context, retain the KV cache, and then generate any number of 256-token proof windows at approximately linear incremental cost in the number of generated tokens. v1 therefore adjusts for model compute, not for every request's context economics. This is less precise than ideal metering, but it is more robust as a consensus rule.

iv. Measuring F — sparsity- and dtype-aware FLOP counting

For a single forward pass on a synthetic batch of size B and length T, every linear layer ℓ with input dimension d_{in} and output dimension d_{out} contributes

$$F_{\text{lin}}(\ell) = 2\, B\, T\, d_{\text{in}}^{(\ell)}\, d_{\text{out}}^{(\ell)} \cdot s_{\text{dt}}(\ell) \cdot (1 - \sigma(\ell)),$$

and every multi-head attention with h heads of dimension d_h contributes

$$F_{\text{attn}}(\ell) = (\, 2\, B\, h\, T^2\, d_h \,+\, B\, h\, T^2 \,+\, 2\, B\, h\, T^2\, d_h \,) \cdot s_{\text{dt}}(\ell),$$

corresponding respectively to the $QK^\top$ score matrix, the in-place softmax, and the attention-weighted value reduction. The dtype scale s_{dt} takes the canonical values 1 (fp32), $\frac{1}{2}$ (fp16, bf16), $\frac{1}{4}$ (int8) and $\frac{1}{8}$ (fp8), reflecting the empirical cost ratios on the dominant accelerator architectures in v1. The sparsity cor-

rection $\sigma(\ell) \in [0, 1]$ is the empirical zero-weight fraction of the layer's weight tensor — a layer that is genuinely sparse at the weight level performs proportionally fewer multiply-adds in practice. Total measured per-token FLOPs is $\hat{F} = T^{-1} \sum_\ell (F_{\text{lin}}(\ell) + F_{\text{attn}}(\ell))$.

The same forward pass also tallies, for each layer, the fraction of output positions with at least one non-zero post-activation coordinate. The aggregate active ratio $\alpha = \frac{\sum \text{activetokens}}{\sum \text{tokens}}$ is reported alongside F. An α near 1 indicates a fully-dense model; α measurably below 1 indicates routed sparsity, in which case F is interpreted as an *expected* FLOP cost averaged over routes.

v. Auxiliary checks

The audit additionally produces three auxiliary statistics that do not participate in $D^\star$ but are reported alongside it.

Saliency by occlusion. For a fixed bank of P random contexts, the audit computes a baseline next-token top-50 softmax distribution $q^{(i)}$. Then, for each parameter matrix $\mathbf{W}$, the audit zeroes contiguous chunks of rows in turn, recomputes the perturbed top-50 distribution $\tilde{q}^{(i)}$, and aggregates the mean total-variation shift

$$\Delta = \frac{1}{P} \sum_{i=1}^{P} \| q^{(i)} - \tilde{q}^{(i)} \|_1 .$$

A chunk is declared *salient* if $\Delta \geq 0.05$, and the salient parameter count is

$$n_{\text{salient}} = \sum_{\mathbf{W},\ \text{chunk}} |\text{chunk}| \cdot \mathbf{1}\{\Delta \geq 0.05\}.$$

Validity probes. Three KL-style statistics are computed against a fixed bank of natural-prose passages tokenised under the model's own tokenizer: a between-passage input-sensitivity KL, a single-token perturbation KL, and a permutation-perplexity ratio $R_\pi = \exp \text{CE}(f_\theta(A_\pi)) / \exp \text{CE}(f_\theta(A))$. A genuine LM exhibits non-trivial input sensitivity, mild but non-zero single-token sensitivity, and $\mathbb{E}[R_\pi] > 1$ 1 regardless of model quality.

File-size sanity. Given n_{salient} and the bits-per-weight b inferred from the in-memory dtype, the expected on-disk size is

$$S_{\text{disk}} \approx n_{\text{salient}} \cdot \frac{b}{8}.$$

The reported ratio guards against weight stripping or model substitution at distribution time.

vi. Registration threshold and implicit hashpower support

The model audit is *not* an oracle and is *not* on the critical path of ordinary block validation. In the shipping inception design it is an advisory input to model validation and operator review. The on-chain registry then records support through deposit and commit transactions:

1. **Deposit.** The registrant locks the model registration deposit and publishes the model metadata: name, commit, claimed difficulty, CID, and auxiliary metadata. The chain stores the model in a pending state and schedules the registration outcome after the registration window.
2. **Validation and commits.** Miners can create model commit transactions during the window only for the exact metadata already deposited. Local mining policy includes successful commits only after the validation service has returned `Model_OK` for the model.

3. **Threshold.** At mainnet inception the registration window is 100 blocks and the success threshold is 50 commit transactions. If the model has at least 50 successful commits when the scheduled outcome is evaluated, it becomes `Registered`; otherwise it becomes `Locked` and its registration deposit enters the burn/reclaim path. The chain counts at most one successful commit for the same model in a block, so this is a 50-of-100 block-slot threshold rather than an unbounded transaction-count threshold.

Hashpower is therefore represented implicitly, not by an explicit commit-reveal vote over D_i. Miners express support by mining commit transactions into the registration window. A model that cannot attract enough block producers to include its commits does not become a usable mining model. [1]

vii. Public pre-alert and challenge discussions

TensorCash ships a public, off-chain discussion surface in the Qt model page so that model admission is not a blind mempool race. The Discussion tab exposes two Nostr-backed thread types:

- `model_prealert` — used before or during model registration so the community can inspect the proposed model identifier, commit, CID, claimed difficulty, audit reports, and monitoring expectations before commits accumulate on-chain.
- `model_challenge` — used once a registered model is being questioned, so challengers and operators can coordinate evidence before and during the challenge window.

For a model pre-alert the Qt wallet derives the scope hash from `HashSHA256(model_name, model_commit)`. For a challenge discussion the Qt wallet requires the scope hash to resolve to an existing model hash in ModelDB, so readers can connect the thread to the registered model being monitored.

Each post is a Nostr kind-8322 event tagged with the thread scope, the network, and the human-readable `model_name@commit_id` identifier when available. The Qt wallet attaches a BIP-322 ownership proof automatically from a confirmed UTXO that meets the chosen stake threshold. The canonical signed message is

```
TENSORCASH_DISCUSS:v1:<network>:<scope_type>:<scope_id>:<nostr_pubkey>:<expiry_height>
```

The proof binds the post to the network, the exact discussion scope, the author's Nostr pseudonym, and an expiry height. On refresh, the node rechecks the BIP-322 signature, the backing UTXO, the active chain, the scope, the Nostr author binding, and the expiry height. These discussion threads are not consensus objects; they are a public coordination layer for the community to review, monitor, and challenge models before hashpower support or challenge commits are committed on-chain.

viii. Challenge phase

Registration is not the end of model accountability. A registered model can be challenged by submitting a challenge deposit against a specific block mined under that model. The challenged block identifies the model hash; the model must still be registered, and only one active challenge window per model is accepted.

1. In the simple independent-block model where a fraction h of block production includes a valid commit for a candidate model, the probability of registration is $\sum_{k=50}^{100} \binom{100}{k} h^k (1-h)^{100-k}$. The statistical boundary is centred at $h = 0.5$: for example, $h = 0.45$ gives about an 18% chance, $h = 0.55$ about an 86% chance, and $h = 0.60$ about a 98% chance. If miners act as deterministic blocs, controlling 50 of the 100 block slots is the registration threshold.

The challenge transaction locks the challenge deposit, records the challenged block hash in the model record, resets the model's `challenge_commit_count`, and schedules a verdict 100 blocks later. It also asks the validation service to perform challenge review. Challenge review has the same operator-gated shape as model review: remote or local validation prepares an audit report, then the operator resolves the challenge as `Challenge_OK` or `Challenge_Fail`.

During the active window, miners may include challenge commit transactions for the challenged model. Local mining policy includes those commits only after the challenge has `Challenge_OK`. Consensus accounting itself is commit-count based: each valid challenge commit increments the model's `challenge_commit_count`, and 50 commits in the 100-block window satisfy the challenge threshold. As with registration commits, only one challenge commit for the same model may count in a single block. Once the threshold is reached the model is marked `Banned`; at the scheduled verdict the challenge state is finalised, the registered model's deposit is burned, and the challenged block's contribution to descendant chain work is removed so the node can select a chain that does not rely on the invalidated proof. If the threshold is not met, the model remains registered and the challenge deposit is burned.

IV. Per-Block Proof Verification

i. The Three-Tier Verification Ladder

Once a model is registered with a difficulty D, every block produced under that model is accompanied by a proof object. Verification of a proof is the operation that we now specify. We organise it as a ladder:

$$\text{Quick} \implies \text{Smell} \implies \text{Full}.$$

Each stage is strictly more selective than the previous. We assign each stage a **gating role** in the network protocol:

Stage	Cost	Network gate
Quick	hashes, table look-ups	A failed Quick stops the proof at the receiving node — the header is **not propagated**, no further work is performed, and no peer is told about the proof.
Smell	$O(W)$ tabular	A failed Smell allows the receiving node to decide *not* to gossip the header to its peers. Peers that have not seen the header therefore see no version of the chain extending through this proof.
Full	teacher-forced window replay	A passed Full allows **block acceptance** and a **chain-tip update**. A failed Full leaves the node's chain tip unchanged.

Quick is therefore a *safety* gate (cheap rejection of obviously broken proofs), Smell is a *liveness* gate (preventing distributionally implausible proofs from polluting the gossip layer), and Full is the *correctness* gate (the only stage entitled to extend the canonical chain).

The cost asymmetry with mining is central. Mining is search: a miner may need to generate many 256-token candidate windows before one boundary hash lands below the adjusted target, and each candidate window is sampled through 256 autoregressive decode steps because the next sampler draw depends on the previous sampled tokens. Full verification is not search. The verifier receives one claimed prompt-plus-window transcript and can replay the core model computation as a teacher-forced full-context pass over the known prompt and 256-token window, producing the logits needed for all positions without walking the miner's search tree. v1 may add auxiliary batched/noisy passes for calibration, but the verifier checks one candidate proof; it does not reproduce the miner's search space.

ii. Quick Verification

The Quick stage performs three independent checks: cryptographic block sanity, hyperparameter eligibility, and per-step sampler consistency. It performs no model forward pass.

Deterministic sampler transcript

The implementation uses the same transcript grammar in two places, but with different roles. Per-step hashes drive token sampling. A separate window-boundary hash is the proof hash whose first four bytes become the block header nonce.

For ordinary sampling step t inside the proof window, define

$$C_t = \mathrm{tail}_{256}(\mathrm{prompttokens} \parallel \tau_0 \parallel \cdots \parallel \tau_{t-1}),$$

left-padded with zero token IDs to exactly 256 positions. Each token ID is encoded as an eight-byte little-endian integer. The sampler message is

$$M_t = \mathrm{header\_prefix} \parallel \mathrm{VDF} \parallel \mathrm{u32le(tick)} \parallel \mathrm{u32le}(t) \parallel \mathrm{i64le}(C_t[0]) \parallel \cdots \parallel \mathrm{i64le}(C_t[255]) \parallel \mathrm{precision\_tag}.$$

The miner computes

$$H_t = \mathrm{SHA256}(M_t), \qquad u_t = \frac{\mathrm{LE32}(H_t[0:4])}{2^{32}} \in [0, 1).$$

The proof stores the float u_t, not the full per-step digest H_t. The verifier recomputes u_t independently for $t = 0, \ldots, W-1$ and asserts $|u_t - u_t^{\mathrm{claim}}| \le 10^{-7}$ before checking that the claimed token lies in the corresponding sampler CDF bin. The first four bytes of these per-step hashes are sampler entropy only; they are not block-header nonces.

At the 256-token solution boundary the miner computes one more transcript hash from the completed rolling window,

$$C_\star = \mathrm{tail}_{256}(\mathrm{prompttokens} \parallel \tau_0 \parallel \cdots \parallel \tau_{W-1}),$$

using the same message layout but with step index 0:

$$H_\star = \mathrm{SHA256}(\mathrm{header\_prefix} \parallel \mathrm{VDF} \parallel \mathrm{u32le(tick)} \parallel \mathrm{u32le(0)} \parallel \mathrm{i64le}(C_\star[0]) \parallel \cdots \parallel \mathrm{i64le}(C_\star[255]) \parallel \mathrm{precision\_tag}).$$

The proof field `hash` is $H_\star$. The block nonce is $\mathrm{LE32}(H_\star[0:4])$, serialized into the header as those first four digest bytes. This is the only place where the first four bytes are used as a header nonce.

This does not give the miner a separate 32-bit search shortcut. The nonce is not an independent field the miner can grind directly: it is valid only if the full 32-byte $H_\star$ equals the verifier's recomputation from the header prefix, VDF, tick, completed token window, and precision tag. Changing the nonce requires changing that transcript, which means changing the block template/VDF/tick/precision or producing a different sampled trajectory. The rolling context prevents pre-mining of the trajectory from the header and VDF alone, because H_t for $t > 0$ is not knowable until the final token in the proof window has been sampled.

Block sanity

The verifier checks the VDF $\mathrm{Verify}(\mathrm{blockhash}, \mathrm{vdf}, \mathrm{tick}) = \mathrm{True}$ against the chain's discriminant, recomputes $H_\star$, checks that it equals the proof's `hash` field, takes `hash[0:4]` as the header nonce, and checks the proof-of-work inequality

$$\mathrm{SHA256}(\mathrm{SHA256}(\mathrm{header} \parallel \mathrm{nonce})) < T_{\mathrm{adj}}.$$

The first guarantees the timing/ordering scaffolding; the second guarantees the underlying hash PoW.

Header flags and advisory gossip reconciliation

TensorCash headers carry one intentionally malleable byte after the consensus hash preimage. In the current C++ header layout, `GetHash()` hashes `nVersion`, `hashPrevBlock`, `hashMerkleRoot`, `nTime`, `nAdjBits`, `nNonce`, `nBits`, and `hashPoW` — 116 serialized bytes in v1 — and deliberately excludes the trailing `flags` byte. Full header serialization still carries that byte on the wire, but it is not part of block identity, chain work, or proof-of-work.

The excluded byte is used as a gossip-layer advisory channel for Full verification results. When a node responds to `getheaders`, it serializes ordinary block headers but sets `header.flags` to its local `GetOwnFullStatus(header.GetHash())`. A receiving node records recognized values — `Full_Green`, `Full_Amber`, or `Full_Red` — as that peer's advisory status for the canonical block hash. Unrecognized values, including `Not_Checked`, are ignored for this purpose.

This design is conscious malleability, not nonce malleability. Multiple peers may relay the same canonical block hash with different trailing flag bytes because they have reached different local Full-verification verdicts. Consensus and PoW checks ignore the byte; the validation API uses it only to poll peer views during AMBER/RED resolution. In particular, an AMBER flow can ask peers for headers ending at the candidate block and use their returned flag bytes as an advisory RED/AMBER/GREEN sample. That makes the existing `getheaders` path double as a lightweight reconciliation mechanism: nodes can discover whether a borderline proof is a local verifier anomaly or a broader network concern before finalising the result, reducing the risk of verifier noise causing unnecessary network partition.

Sampler consistency

Given the persisted top-K logits $\tilde{L}_t$ and indices S_t, the verifier reconstructs the sampling distribution that the miner *must have* used, by following the canonical sampler exactly: temperature scaling, repetition penalty, top-k truncation, top-p truncation (over the finite proof support, with token id ascending as the canonical secondary sort key), and softmax normalisation,

$$\pi_{t,i} = \frac{\exp(\ell_i)}{\sum_j \exp(\ell_j)},$$

where ℓ_i are the truncated logits. The verifier then constructs an **ID-sorted CDF**

$$F_t(m) = \sum_{j=1}^{m} \pi_{t,(j)},$$

with $\pi_{t,(j)}$ ordered by token ID ascending, and asserts that the miner's chosen token τ_t — which lands at some position m_t in this ordering — has u_t inside its bin:

$$F_t(m_t - 1) - \varepsilon < u_t \leq F_t(m_t) + \varepsilon.$$

ID ordering is essential. Floating-point summation is non-associative, so a CDF built in *probability* order may disagree across platforms in the last bit; a CDF built in *token-ID* order is purely combinatorial once the truncated probabilities are fixed, and is reproducible across all platforms to within rounding tolerance ε.

Quick gate: header propagation

If any of the three checks above fails, the proof is rejected at the receiving node. The corresponding header is **not propagated** to peers.

iii. Smell Test

The smell test asks whether the persisted top-50 distribution of the proof is consistent with random Monte-Carlo runs of the same model. It runs in $O(W)$ on the persisted logits alone, requires no recomputation of the model, and provides a fingerprint check that is hard to spoof without access to the actual weights.

At mainnet inception, the fingerprint F is verifier-local state: it is loaded from the node's cache when available, or calibrated from the verifier's local copy of the model. The fingerprint is not a consensus object and is not shared on-chain in v1. If a verifier has no fingerprint available, the shipping implementation treats Smell as skipped rather than as a hard failure; future versions may make fingerprints reproducible or registry-published. We describe the three Mahalanobis components first and the calibration last.

Inert-token frequency test

Let $S \subset \{1, \ldots, V\}$ be a fixed set of *inert tokens* with cardinality K_S — those tokens whose probability of appearing in a random next-step top-50 has empirical mean in $(0.001, 0.05)$ and whose Bernoulli information $\mu(1-\mu)$ is highest. Let $\bar{c} \in \mathbb{R}^{K_S}$ be the empirical mean, and Σ_c the Ledoit–Wolf-shrunk covariance, of the inert-token count vector taken over chunks of length 256 of calibration runs.

For a candidate proof, define $c \in \mathbb{R}^{K_S}$ as the vector of counts of each inert token in the proof's $W \times 50$ index matrix. The test statistic is

$$M^2_{\text{freq}} = (c - \bar{c})^\top \Sigma_c^{-1} (c - \bar{c}) \sim \chi^2_{K_S},$$

with $p_{\text{freq}} = 1 - F_{\chi^2_{K_S}}(M^2_{\text{freq}})$.

Token marginal frequencies are a robust invariant of the unembedding matrix and the residual stream norm: any plausible substitute model — a fine-tune, a related-but-different checkpoint, a stripped distillation — shifts the joint distribution of S in a multivariate way that is detectable in K_S dimensions. Ledoit–Wolf shrinkage guarantees positive-definiteness of Σ_c at the calibration sample sizes used.

49-gap mean test

For each step, define the 49-vector of consecutive top-50 logit gaps,

$$g_t = (\tilde{L}_{t,(1)} - \tilde{L}_{t,(2)}, \ \ldots, \ \tilde{L}_{t,(49)} - \tilde{L}_{t,(50)}),$$

and the proof statistic $\hat{g} = \frac{1}{W}\sum_t g_t$. With calibration $(\bar{g}, \Sigma_g)$ — accumulated by an online Welford estimator on chunk means, Tikhonov-regularised by $5 \cdot 10^{-6}\, I$ to ensure non-singularity — the test statistic is

$$M^2_{\text{gap}} = (\hat{g} - \bar{g})^\top \Sigma_g^{-1} (\hat{g} - \bar{g}) \sim \chi^2_{49}.$$

The shape of the logit decay is a near-linear function of the unembedding matrix and the residual-stream scale. Tampering with the top-50 logits without the actual weights is detectable in the 49 gap coordinates because gap shape varies smoothly with model architecture and identity.

Cosine-dispersion test

Reduce the model's input embedding matrix to its first 16 principal components, $\mathbf{E} \in \mathbb{R}^{V \times 16}$. For each step compute the mean upper-triangular cosine similarity over the 50 top-token embeddings,

$$\bar{s}_t = \binom{50}{2}^{-1} \sum_{1 \le i < j \le 50} \frac{\langle \widehat{\mathbf{E}}_{S_{t,i}}, \widehat{\mathbf{E}}_{S_{t,j}} \rangle}{\|\widehat{\mathbf{E}}_{S_{t,i}}\| \, \|\widehat{\mathbf{E}}_{S_{t,j}}\|},$$

and the proof statistic $\bar{s} = \frac{1}{W}\sum_t s_t$. The Fisher transform $z = \operatorname{arctanh}(2\bar{s} - 1)$ is approximately normal under H_0, so the test statistic is

$$M^2_{\cos} = \left(\frac{z - \mu_z}{\sigma_z}\right)^2 \sim \chi^2_1, \qquad p_{\cos} = 1 - F_{\chi^2_1}(M^2_{\cos}),$$

with (μ_z, σ_z) from calibration.

This catches an attacker who synthesises top-50 indices whose marginal frequencies are calibrated correctly but whose embedding-space clustering is not. A real LM's top-50 at any given step lives in a small cone of the embedding space; an adversarial sample drawn from a model with a different unembedding will break the cone structure even when frequencies match.

Joint decision

The smell test passes iff each component is below its calibrated 99th percentile:

$$(M_{\text{freq}} \le \theta_{\text{freq}}) \wedge (M_{\text{gap}} \le \theta_{\text{gap}}) \wedge (M_{\cos} \le \theta_{\cos}).$$

The thresholds are taken as empirical 99th percentiles of the joint calibration distribution rather than as analytical χ^2 critical values, because at chunk sizes that are practical to calibrate, the analytical asymptotics are not yet tight.

Smell gate: header gossiping

A proof that passes Quick but fails Smell is **not gossiped**. Its header is allowed to remain on the receiving node (so that the node may re-evaluate later if its calibration changes), but the node will refuse to forward the header to its peers, ensuring that distributionally implausible proofs do not propagate through the gossip layer.

iv. Full Verification

This is the most expensive stage. The verifier re-runs the model and asks whether the proof's logits sit inside the cone of acceptable floating-point noise around its own. The technical heart consists of three intertwined ideas:

1. **Snapping to precision** — observation and null are both projected onto the dtype lattice declared by the proof, so that the test statistic compares like with like.
2. **Monte-Carlo arithmetic (MCA)** — controlled Gaussian noise is injected into selected floating-point operations during the verifier's forward pass, used to *expand* the empirical noise envelope so it covers legitimate cross-platform divergence.
3. **Mahalanobis test in 74 dimensions** — combining 70 quantised per-logit deltas and 4 bucket-mean deltas, with a non-diagonal covariance that captures cross-logit correlations introduced by shared accumulation paths.

The 74-dimensional observation

For step t, let $L^B_t \in \mathbb{R}^V$ be the verifier's full-vocab logits ("platform B"), and let $\tilde{L}^A_t \in \mathbb{R}^{70}$ be the miner's persisted top-70 logits ("platform A"), evaluated on the indices S_t. Define the global mean offset

$$\Delta\mu_t = \operatorname{mean}(L^A_t) - \operatorname{mean}(L^B_t).$$

This is a benign per-step shift: softmax is invariant to additive constants, so any global bias between the two platforms cancels under sampling. Subtracting it gives the centred logit difference

$$d_t = \tilde{L}_t^A - L_t^B[S_t] - \Delta\mu_t \in \mathbb{R}^{70}.$$

For each bucket $[a, b) \in B$, let $\mu_t^{A,[a,b)}, \mu_t^{B,[a,b)}$ denote the means of the *full* sorted logit vector over that bucket on each platform. The bucket-mean deltas are

$$m_t = \mu_t^A - \mu_t^B - \Delta\mu_t \in \mathbb{R}^4.$$

The 74-dimensional observation is $v_t = (d_t,\ m_t)$. The four bucket coordinates summarise tail mass: a tampered proof that perfectly matches the top-70 may still be detectable in the bulk of the distribution through these means.

Snapping to the precision grid

Both platforms produce values that already live on a finite floating-point lattice. Comparing a continuous Gaussian sample to an unsnapped real value distorts the test's tail and inflates Type-I error. We therefore snap both the observation and the null to the same lattice. Define the per-coordinate ulp width

$$u_t^{\text{ulp}} = 2\ \text{ulp}(L_t^B[S_t];\ \text{dt}),$$

(the factor 2 accounts for representable-value spacing across binade boundaries), and let

$$\widehat{L}_t^B = \text{snap}(L_t^B[S_t];\ u_t^{\text{ulp}})$$

be the snapped base point. The discrete component of the observation is

$$D_t^{\text{obs}} = \tilde{L}_t^A - \widehat{L}_t^B \in \mathbb{R}^{70}.$$

If the miner's persisted logits do not themselves live on the dtype grid declared in the proof — i.e. if $|\text{snap}(\tilde{L}_t^A) - \tilde{L}_t^A|$ exceeds round-off tolerance — the Full stage rejects immediately. This is the *grid-snap fail* gate.

The error covariance

The covariance of platform noise is composed of three contributions taken pointwise as a maximum, then assembled into a non-diagonal matrix.

ULP-induced standard deviation. A classical Wilkinson bound gives the accumulated rounding error of k sequential floating-point sums as $O(\sqrt{k}\ \text{ulp})$. Mapping the peak bound to a root-mean-square via the uniform-jitter factor $1/\sqrt 6$,

$$\sigma_{\text{ulp}}(x) = \text{ulp}(x) \cdot \sqrt{k/6},$$

with $k = 5$ in v1 (a deliberately conservative effective accumulation depth). This is computed both per-coordinate and as a global average over the full sorted logit vector; the elementwise maximum of the two prevents a single very small logit from dragging the entire σ floor down.

Magnitude floor. A small fraction of the spread of the sorted logit vector, $\sigma_{\text{floor}} = \min(0.005 \cdot \text{std}(L_t^B),\ 0.1)$, guards against exact-equality tail events where two coordinates collide to the same representable value.

Empirical covariance adjuster. The shipping mainnet-inception verifier uses the batched `v2` path. It computes the whole proof window under a teacher-forced single-batch baseline over the full known context and 256-token window, a family of MCA-noisy candidate batch sizes, and a single-batch attention-noise run. A vectorised estimator then compares the baseline, the largest candidate batch, and the attention-

noise run across the window to produce the covariance floor, rank-swap calibration, and platform-shift calibration. This captures noise sources that the ULP-and-magnitude analysis above does not predict — for example SDPA-vs-eager kernel divergence on long sequences. A legacy warm-up path over the first 40 steps remains available, but it is not the default verifier path.

The final per-coordinate $\sigma_t \in \mathbb{R}^{74}$ is

$$\sigma_t = \max(\sigma_{\text{ulp}}, \sigma_{\text{floor}}, \sigma_{\text{cov-adj}}).$$

For the four bucket coordinates we additionally rescale by $1/\sqrt{N_b/4}N_b$ the bucket size, recovering the standard $1/\sqrt{n}$ shrinkage of the bucket mean's standard error.

The full covariance is then assembled with a fixed cross-logit correlation matrix $C \in \mathbb{R}^{74\times 74}$, calibrated off-line on a reference model:

$$\Sigma_t = \text{diag}(\sigma_t)\, C\, \text{diag}(\sigma_t).$$

Logit errors are not coordinate-wise independent: two top-5 positions that share an accumulator path in a fused matmul share the same rounding error to first order, and modelling that correlation closes a leak diagonal-Σ tests have. Treating fluctuations as independent makes correlated jitter look more anomalous than it is and inflates the false rejection rate.

Monte-Carlo arithmetic

MCA is the deliberate injection of bounded random noise into selected floating-point operations during the verifier's own forward pass. It is applied to two operation classes: dense linear operations while executing attention modules, and the scaled-dot-product attention output. This covers the usual W_Q, W_K, W_V, W_O projection path while keeping the MLP path deterministic in the standard transformer implementation; fallback attention hooks cover non-SDPA attention paths. For an output tensor y at one of these operations, the perturbed output is

$$\tilde{y} = y + |y| \odot \xi, \qquad \xi \sim N(0,\ (k\,\varepsilon_{\text{dt}})^2),$$

i.e. a *relative* Gaussian whose scale is k times the per-dtype machine epsilon. The choices in v1 are $\varepsilon_{\text{fp16}} = 4.88 \cdot 10^{-4}$, $\varepsilon_{\text{bf16}} = 7.81 \cdot 10^{-3}$, $k_{\text{lin}} = 1.5$, $k_{\text{attn}} = 8.0$. The attention scaling factor grows with the sequence length to track the $O(\sqrt{T}\,\varepsilon)$ behaviour of T-element softmax-weighted reductions:

$$k_{\text{attn}}^{\text{eff}}(T) = k_{\text{attn}} \cdot \sqrt{\max(T,1)/T_{\text{piv}}}, \qquad T_{\text{piv}} = 512.$$

MCA is a *calibration tool* for the null. Its purpose is not to corrupt the verifier's signal but to widen $\widehat{\Sigma}_{\text{cov-adj}}$ so that the verifier's empirical noise envelope covers honest cross-platform divergence — different GPUs, different batch sizes, different attention backends. The batched verifier folds MCA-noisy, attention-noisy, and MCA-quiet runs into the same covariance model, so the floor σ_t used by the error-covariance model accounts for the union of these noise sources.

The Mahalanobis statistic and bootstrapped null

The observed test statistic is the squared Mahalanobis distance,

$$T_t^{\text{obs}} = v_t^{\top} \Sigma_t^{-1} v_t.$$

The null distribution of T_t under H_0 is *not* a closed-form χ^2_{74} — the discrete component is snapped, so its marginal is a quantised Gaussian. We therefore generate the null empirically. Let $L_t = \mathrm{chol}(\Sigma_t)$ and draw $z^{(b)} \sim N(0, I_{74})$ for $b = 1, \ldots, B$ (default $B = 15{,}000$). Set $\eta^{(b)} = L_t z^{(b)}$ and form

$$x^{(b)} = L_t^B[S_t] - \Delta\mu_t + \eta^{(b)}_{1:70},$$

$$D^{(b)}_{\text{null}} = \widehat{L}_t^B - \mathrm{snap}(x^{(b)};\, u_t^{\text{ulp}}), \qquad C^{(b)}_{\text{null}} = \eta^{(b)}_{71:74},$$

$$V^{(b)}_{\text{null}} = (D^{(b)}_{\text{null}},\, C^{(b)}_{\text{null}}).$$

The empirical p-value is

$$\hat{p}_t = \frac{1}{B} \sum_{b=1}^{B} \mathbf{1}\{ V^{(b)\top}_{\text{null}} \Sigma_t^{-1} V^{(b)}_{\text{null}} \geq T_t^{\text{obs}} \}.$$

If $\hat{p}_t < 0.01$ the verifier draws an additional $10B$ bootstrap samples, refining the granularity of the tail estimate; if still $\hat{p}_t < 0.001$, it draws a further $2B$ samples under an inflated covariance $1.2\,\Sigma_t$, as a final paranoia step. None of these re-bootstraps re-orders observations; they only sharpen the resolution of the empirical CDF in the tail.

The reason for snapping inside the null is calibration. The discrete component $D_{\text{null}}{}^{(b)} = \hat{L}_t^B - \mathrm{snap}(\hat{L}_t^B + \eta)$ is the value the null hypothesis would have produced under the same quantisation. A test that compares an unquantised η to a quantised D^{obs} is mismatched and biased.

Adaptive per-step refinement

For each step t, the default batched verifier evaluates the baseline, the MCA candidate set $\{2, 5, 10, 20\}$, and the attention-noise candidate, then selects the candidate with the largest Mahalanobis p-value for the main anomaly stream. For the sampling-noise and platform-shift streams it uses the candidate with the smallest absolute residual. This is a full-window vectorised selection, not a serial retry loop. The legacy verifier can still run a serial adaptive refinement path, but that is not the inception default.

Per-step $\hat{p}_t$ are not themselves the verdict — they are aggregated across the proof window in the aggregate verdict section.

Full gate: block acceptance and chain-tip update

A proof that passes the Full stage with verdict GREEN entitles the receiving node to accept the block and update its chain tip. Verdicts of AMBER and RED are described next.

v. Aggregate Verdicts: RED, AMBER, GREEN

The Full stage produces, for the full proof window, four streams of statistics:

- per-step Mahalanobis p-values $\{\hat{p}_t\}$,
- per-step rank-swap counts $\{R_t\}$ between platforms A and B over the top-50 indices,
- per-step sampling-noise residuals $\{\eta_t\}$ — the difference between the proof's chosen-token CDF position and the verifier's reconstruction,
- per-step platform shifts $\{\Delta\mu_t\}$.

For each stream we apply fraction gates of the form (q, t): no more than a fraction t of observations may fall on the wrong side of threshold q. Lower-tail gates are used for p-values; upper-tail two-sided gates are used for sampling noise and platform shift. The shipping table has RED and AMBER tiers for the p-value stream, RED-only tiers for sampling noise and platform shift, and a separate RED/AMBER rank-swap gate.

Stream	Fraction gate (q, t)
p-values (RED)	$(0.5, 0.75),\ (0.05, 0.10),\ (0.01, 0.02),\ (0.001, 1/256)$
p-values (AMBER)	$(0.5, 0.60),\ (0.05, 0.06),\ (0.01, 0.01),\ (0.001, 0)$
Sampling noise (RED)	$(0.005, 0.50),\ (0.01, 0.10),\ (0.03, 0.01),\ (0.05, 2/256),\ (0.15, 1/256)$
Platform shift (RED)	$(\sigma, 0.40),\ (2\sigma, 0.20),\ (5\sigma, 0.10),\ (10\sigma, 2/256),\ (20\sigma, 1/256)$

The rank-swap stream is treated separately. Calibration values $\{R_t^{\text{calib}}\}$ are Winsorized at the 99th percentile, fit to a Negative Binomial via method of moments with floors $r \geq 0.5$ and $p \geq 10^{-3}$, and used to compute per-step survival probabilities s_t. These are clipped to $[10^{-12}, 1]$ and combined via Fisher's method,

$$X^2 = -2\sum_t \log s_t \sim \chi^2_{2W}, \qquad p_{\text{rank}} = 1 - F_{\chi^2_{2W}}(X^2).$$

The rank-swap test triggers RED at $p_{\text{rank}} < 5 \cdot 10^{-4}$ and AMBER at $p_{\text{rank}} < 10^{-3}$.

The verdict is then assigned according to the rule: **RED** if any RED gate trips or the grid-snap check fails; **AMBER** if no RED gate trips but at least one AMBER gate does; **GREEN** otherwise.

Operational meaning of each verdict

GREEN — the proof is statistically indistinguishable from a faithful execution of the declared model under any platform configuration in the v1 noise envelope. The block is accepted and the chain tip advanced.

AMBER — the proof is statistically *inconvenient* but not disqualifying. At least one soft gate has fired, but no severe gate has fired. AMBER is consensus-neutral in v1: a node that observed AMBER is permitted to *re-enqueue* the proof for a fresh Full execution, on the chance that the failure was caused by transient noise on the verifier side (e.g. a co-tenant process consuming GPU bandwidth, an opportunistic kernel selection). The block is *not* accepted on the basis of a single AMBER outcome, but a node observing repeated AMBERs across independent Full runs escalates to RED. The exact escalation policy is deliberately left as an open research direction; v1 admits any policy that reduces to GREEN or RED on retry, provided the policy is monotone (more AMBERs cannot overturn a previous RED).

RED — the proof is statistically incompatible with the declared model, the declared precision, or the canonical sampler. The block is rejected, the chain tip is unchanged, and the receiving node is permitted to penalise the source peer in its gossip-layer reputation score. RED is a terminal verdict at the v1 verifier; future versions may add a recovery path conditioned on, for example, an attestation by a third-party verifier, but no such path exists in v1.

These verdicts are the *only* outcomes a Full stage may produce. The verdict surface — RED, AMBER, GREEN — and the gating roles they control are intended to be stable across verifier versions. The internals of the gates are not. A successor verifier V_{n+1} may re-derive its acceptance region from first principles, change the covariance model, replace the bootstrap with a closed-form null, or add entirely new

statistics; v1 makes no claim that its internal calibration is final, and successor versions are not bound by v1's individual verdicts. The protocol's forward-compatibility surface is the *shape* of the decision (a tiered ladder gating propagation, gossiping, and chain acceptance), not its v1 implementation.

V. Attack Surface Covered by v1

The main whitepaper discusses the economic and architectural attack surface. This section records how the v1 verifier maps those attacks to concrete gates.

Premine and prompt-bank reuse. A miner may cache prompts, completions, or partial transcripts hoping they later become valid. Quick binds each sampler draw and boundary proof hash to the current header prefix, VDF output, tick, rolling token context, and precision tag. A banked prompt may still be a valid input, but its mining value cannot be known without re-executing the transcript under the actual chain state.

Fan-out, early exit, and nonce shortcutting. A miner may try to branch cheaply, stop early, or treat the four-byte header nonce as an independent search field. Quick prevents this: the block nonce is accepted only as `hash[0:4]` from the full recomputed boundary digest, and the digest is defined only after the 256-token window exists. Partial paths do not have an admissible boundary proof.

Low-entropy degeneracy. Greedy or near-deterministic generations reduce the statistical content of a proof and create opportunities for surrogate guessing. The sampling-contract entropy exclusion rejects proof windows whose mean chosen-token probability is too high, keeping accepted proofs in a distributional regime where Smell and Full retain power.

Fake compute and hollow models. A registrant may try to advertise a large model while most of the declared computation is inactive or short-circuited. Model admission addresses this through measured FLOPs per token, sparsity- and dtype-aware accounting, saliency-by-occlusion checks, validity probes, file-size sanity checks, deposit-backed registration, and the 50-of-100 commit threshold. The challenge phase keeps registered models accountable after admission.

Quantization, distillation, and surrogate substitution. A miner may serve a cheaper model or undeclared precision while claiming the registered one. Full verification compares committed top-logit and bucket statistics against a calibrated platform-noise envelope. Honest cross-platform variation should look like noise; model substitution and undeclared quantization create structured shifts.

Speculative filtering. A miner may use a cheap model to guess which prompts are worth running through the registered model. v1 cannot and does not ban all off-path speculation. It limits the benefit by requiring any accepted block to contain a transcript that passes Quick, Smell, and Full for the registered model. Speculation can reduce a miner's private search cost only if it still culminates in real proof-bearing inference.

VDF and selfish-mining context. VDF anchoring and selfish-mining mitigations are chain-level mechanisms rather than Full-verifier statistics. The verifier consumes the VDF output and tick as transcript inputs, while the consensus layer enforces cumulative timing and chain-work rules.

VI. Operating Modes for Verification

The verifier is specified as a protocol function, not as a mandatory deployment topology. Implementations can choose how to run the expensive parts of the ladder according to their operational needs, trust model, and hardware budget.

The most conservative mode is a self-maintained verifier. A mining node, serving node, exchange, or archival node can operate its own model cache, validation service, and Full-verification workers. In this mode the node does not outsource its RED/AMBER/GREEN decision. It pays the operational cost of model hosting and verifier maintenance in exchange for maximum independence.

A lighter mode is delegated Full validation. A node may perform cheap local checks and delegate Full verification to one or more validator services it chooses. This can be appropriate for operators that want to participate in the network without maintaining every model and inference stack locally. Delegation changes the operator's trust assumption; it does not change the proof format, the Quick/Smell/Full ladder, or the meaning of a verdict. A node remains responsible for deciding which validator outputs it is willing to rely on.

A third mode is fleet validation. A miner, API provider, or aggregator running many mining and serving instances can operate a shared Full validator across the fleet. The individual nodes keep their chain and gossip responsibilities, while a hardened verifier service amortizes model loading, calibration cache, GPU scheduling, and challenge investigation across many workers. This is likely to be the natural shape for commercial inference providers and large mining operations.

The reference codebase includes an open-source implementation of a centralized validation API service. It is useful as a concrete inception implementation and as an integration target for nodes, miners, and operators. It should not be read as the only acceptable topology. More commercially hardened, decentralized, federated, audited, or market-based validator implementations would be welcome, provided they preserve the protocol-facing interface: deterministic Quick checks, statistical Full verdicts in {RED, AMBER, GREEN}, and clear operator accountability for delegated decisions.

VII. Open Research Directions

The verifier specified above is an explicitly v1 design. We close the specification with a non-exhaustive list of directions in which v2 and beyond are expected to evolve. None of these threaten the interface contract defined above (a three-tier ladder gating propagation, gossiping, and chain acceptance, with verdicts in {RED, AMBER, GREEN}); each replaces or refines the **internals** of one or more gates, while leaving wallets and miners unaffected by the upgrade.

1. **Closed-form null distributions.** The empirical bootstrap of the Full stage is a deliberate concession to the difficulty of analysing snapped Mahalanobis statistics. A closed-form (or piecewise-closed-form) null would remove the $B = 15{,}000$ Cholesky-multiply per step at negligible cost in calibration accuracy, and would tighten the tail aggregate gates.
2. **Per-architecture correlation matrices.** The fixed cross-logit correlation matrix C in the error-covariance model is calibrated on a reference model and reused. A model-specific or architecture-specific C would yield a tighter Σ_t, increasing the verifier's discriminative power.
3. **Refined low-entropy filter.** The scalar threshold $\bar{p} < 0.925$ in the low-entropy exclusion is a coarse instrument. A per-step entropy floor — possibly derived from a hash of the prompt context — would protect higher-confidence segments without admitting wholly degenerate proofs.
4. **Audit reproducibility.** The model audit mixes randomness (synthetic prompts, random parameter occlusions) with deterministic accounting (FLOPs, on-disk size). A fully reproducible audit protocol — one that can be re-executed by any participant against the registry's published seed — would reduce one of the remaining off-chain trust assumptions in the system.
5. **AMBER re-enqueue policy.** v1 leaves the AMBER escalation policy unspecified beyond monotonicity. A formal treatment of when a node should accept, re-enqueue, or escalate to RED is itself a reinforcement-learning problem on the operator side, and a Byzantine-fault-tolerance problem on the network side.

6. **Compositional verification.** The three tiers as specified are serial. A future version may admit *compositional* gates — for example, a Smell pass that strengthens the prior on Σ_t for the Full stage, allowing tighter gates without retraining the covariance machinery.
7. **Multimodal inputs and outputs.** v1 is written for language-model token streams, but the transcript idea should extend to multimodal models. The preferred research direction is deterministic canonicalisation and bucketing of modality outputs — codec tokens, patches, chunks, or other discrete public representations — rather than exposing model-internal embeddings. Embeddings are continuous, architecture-specific, and often proprietary to the inference stack; deterministic buckets are hashable, portable, and better aligned with the proof interface.

What we ask of v1 is not finality but **adaptability**: that its interface remains intelligible to a v2 author, that its gates are stated explicitly enough to be replaced rather than reverse-engineered, and that its verdicts are interpretable without reading the verifier's source. The mathematics above is offered in that spirit — as the best we know how to write today, and as a substrate on which strictly better systems can be built.