Proof of Inference (PoI)

A consensus-integrated verified-inference layer, native on Waves

Alexander Ivanov

Verified AI inference is folded directly into Waves' Fair Proof-of-Stake [1]. Nodes mine WAVES by forging blocks; running verified inference makes them more likely to forge and additionally pays them TokenZ. With no jobs, the network falls back automatically to pure PoS. This ships into the Waves node as a consensus-level protocol feature (activated via Waves' feature-voting), using the real WAVES token — not a fork.

Token model (two tokens)

WAVES (the native coin): gas, staking, forging, block reward, the opt-in slashable bond, and governance. The security/value asset.

TokenZ (credit): a claim on inference. The work/credit rail.

Circular flow (single faucet, single drain). TokenZ is emitted to WAVES stakers alongside their WAVES rewards — staking yields both WAVES and spendable AI credits. Stakers (and other holders/users) spend TokenZ on inference; it flows to the serving nodes as payment. Nodes earn TokenZ only by being paid with it (recycled, never separately minted) — their real incentive is the WAVES forging boost, denominated in the appreciating security asset, so TokenZ inflation cannot quietly break node economics. One faucet (staker emission), one drain (inference spend → nodes).

Claims must be capacity-backed. TokenZ is a claim on GPU-time. Staker emission is tied to realized network inference capacity, not to stake alone: minting credits faster than nodes can serve issues more claims than compute exists, devaluing the credit or congesting redemption. This is the demand-side mirror of the wash-inference rule. WAVES governance sets the emission policy — the capacity-to-emission ratio and how it tightens over time — but only within a protocol-enforced capacity ceiling, so no vote can mint more credits than the network can serve. Governance chooses the policy; the per-epoch amount is computed automatically from verified capacity.

Value capture to WAVES from inference is wired explicitly (e.g. inference settlement consuming WAVES gas) — minting WAVES via forging does not by itself tie its price to AI usage but should have a positive impact on its price through increased chain utility.

Design principle

Stake (WAVES) is the security floor. Forging requires a minimum staked balance; security never drops below pure Waves PoS.

Verified work is a capped multiplier on forging weight. Inference amplifies a staker's block-production probability; it never substitutes for stake, so a stake-less GPU farm cannot seize consensus.

TEE + proof-of-inference verify the work that earns the multiplier. TEE attests what ran [2, 3] (pins the model, keeps inputs private); random re-execution of past jobs is the backstop if a vendor key leaks.

Ordinary WAVES stakers without GPUs still forge at base weight — they are the baseline validator set; GPU operators who also bond WAVES earn the boost on top.

How it integrates with Waves FPoS

Waves FPoS block delay is approximately [1]:

```
T_i = T_min + C1 · ln( 1 − C2 · ln(X_n / X_max) / (b_i · A_n) )
```

where b_i is the generating balance, A_n is baseTarget (~60s target), and X_n is the VRF hit (first bytes of a VRF output, unpredictable until the generation signature is published because it needs the generator's private key — this is what makes FPoS grinding-resistant). Smaller T_i → forge sooner.

The only modification is to the balance term:

```
b_eff(i) = b_i · (1 + min(workBoost_i, B_max)),   require b_i ≥ b_min
```

workBoost_i = node i's normalized, finalized verified-work over a trailing window.

B_max = cap on the multiplier (e.g. ×2–3) — bounds centralization pressure.

No jobs → workBoost_i = 0 → b_eff = b_i → vanilla FPoS. Automatic fallback, no special case.

The VRF hit X_n is untouched — work scales the deterministic target only, never the randomness, preserving grinding resistance.

Pre-finalization rule (non-negotiable): work verified in epoch E (attestation checked, spot-check/challenge window closed) is written to a canonical on-chain verified-work balance that boosts forging in epoch E+1. Validators re-derive b_eff from finalized state at validation time — no heavy verification at consensus speed — and work is fixed before the next hit is knowable, so it can't be ground against the draw.

Misbehavior suspends generation, not principal. Waves LPoS in its current form is non-slashable, and this design keeps it that way: no staked or leased WAVES is ever confiscated. Instead, a node proven to have cheated on inference has its generating balance suspended for a penalty period — it keeps every coin, but cannot forge blocks, earn the work boost, or collect verification income while suspended, and its accumulated workBoost is reset to zero and must be rebuilt from scratch on return. The penalty is deliberately reversible and proportionate, which is the right pairing for statistical verification that carries a nonzero false-positive rate: a wrongly suspended honest node recovers, whereas a burned principal could not. Delegators' leased funds are never at risk, so the "leasing is safe" social contract holds even for nodes that serve inference. The rewards a suspended node forfeits during its penalty fund the verifier that caught it and the verification pool, replacing the catch-bounty that a confiscated bond would otherwise provide.

Native implementation. Because the change ships into the Waves node itself, verified-work balances live as a protocol-level state the consensus reads directly (no dApp indirection). A

native secp256r1 verification opcode can be added to move part of the attestation check on-chain — though full X.509 chain validation remains too heavy for block-validation speed, so the E→E+1 finalization lag stays. The feature reaches mainnet through Waves' feature-activation voting, not a code merge alone.

Self-dealing the forging boost

A node can run real, honest inference for itself to inflate its workBoost and capture more WAVES forging share. Verification cannot catch this — the work is genuine; only the demand is fake. It is an economic attack, defended economically. Note it is already bounded: block issuance is fixed, so the boost redistributes a fixed reward pie rather than minting WAVES (zero-sum, bought with real compute, like a PoW miner buying share); and B_max, the per-device cap, and the b_i ≥ b_min floor keep it below the stake-anchored security floor. The defense makes it strictly unprofitable, in three parts:

Per-job economics with compute cost c, fee f, marginal boost value β, burn fraction b: honest serving nets (1−b)f + β − c; self-dealing nets β − c − b·f (the self-paid fee cancels except the burn). Self-dealing is deterred when b > (β − c) / f.

Cap the boost below compute cost (β ≤ c) — the primary lever. If one job's forging boost is worth less than the compute to produce it, work never pays for the boost alone; it only pays when paired with a real external fee. Self-dealing then loses money before any burn, and is dominated by both honest serving and plain staking.

Burn 20–30% of every inference fee, as WAVES buy-back-and-burn. The burn falls on the self-dealer (no external revenue to offset it) far more than on honest nodes (who keep 70–80% plus their external fee), so it discriminates cleanly; it doubles as the WAVES value-capture mechanism. Note the hard limit: if β > f + c, even a 100% burn cannot deter — only the boost cap can — so the burn is a backstop, never a sole guarantee.

Bootstrap window. The boost's marginal value per job is highest when honest demand is low, so self-dealing is most attractive when the network inference feature is young. The sensible approach would be keeping B_max low (or the boost off) until verified external demand crosses a threshold, then raising it.

Participants

Nodes — confidential-compute GPUs that bond WAVES, serve inference, and forge.

Stakers — hold/lease WAVES, forge at base weight (no GPU required), and receive TokenZ credits to spend on inference.

Users — pay TokenZ for answers.

Validators — bonded nodes that re-check others' past work.

To earn the boost, a node (1) registers its GPU's attestable chip identity, (2) bonds the minimum WAVES, and (3) accrues verified work.

Components

1. Identity & registration. On-chain state binds: chip identity ↔ operator wallet ↔ loaded model hash ↔ bonded WAVES. The signing keypair is generated inside the enclave and bound to the attestation, so every response is signed in-enclave over the full request/response transcript — closing the gap where a TEE proves a model is loaded but not that the endpoint routed the request to it. The bound model hash gives model-pinning (anti-bait-and-switch). One attestable chip = one identity, so Sybil attacks cost real GPUs.

2. Work, rewards, and forging weight. Users and stakers spend TokenZ on jobs. A node runs the model in its TEE and returns answer + attestation + trace commitment. Each verification-passed job yields a receipt with a work measure (attested output tokens × model-size class), capped per device. Finalized verified work pays the node in recycled TokenZ (the fee spent by the requester) and feeds workBoost_i for next-epoch forging weight — the node's primary, appreciating reward. A capacity-tied emission throttle sets TokenZ minted to stakers per epoch: WAVES governance sets the policy (capacity-to-emission ratio and its tightening schedule) within a protocol-enforced ceiling, and the amount is computed automatically from realized inference throughput so outstanding credits stay backed by servable compute.

3. Verification (layered defense-in-depth). No single check is trusted; each layer catches what the others miss. A precondition for all of them: verifiable jobs run under declared, fixed decoding parameters (and a committed RNG seed), or there is no clean reference to compare against.

- *Primary, every job — TEE attestation (off-chain).* Validators check the quote against the vendor root and the registered model hash [2, 3]. ~4–8% GPU overhead; frontier-scale. Catches the lazy cheater; provides privacy + model-pinning. Hardware-backed attestation is argued to be the only currently robust defense against silent model substitution [4].
- *Per-output, if the network distributes its own models — watermarking.* A keyed statistical watermark embedded in the logits [7] lets any verifier check the output text alone (no re-execution) and detect a substitute model that lacks the mark. Cheapest first-line check; weaknesses are heavy paraphrasing/editing and a small quality cost.
- *Continuous, per-node — Model Equality Testing [5].* A text-only, distributional two-sample test (MMD [6] with a Hamming kernel + permutation test) compares a node's output stream against a locally-run reference of the true model. Detects systematic substitution, quantization, or fine-tuning from text alone, with no GPU re-execution and a tunable p-value — giving slashing a defensible statistical threshold rather than a subjective verdict. It is a verdict over many jobs, not one, and needs enough samples to separate same-family models.
- *Per-job, sampled — logit spot-check by an attested-verifier committee (the main correctness mechanism).* For each sampled job, several VRF-selected verifiers — each running inside its own attested TEE — independently teacher-force and re-run one (or a few) forward step(s) of the randomly chosen past job, comparing committed logits with nondeterminism-tolerant matching (locality-sensitive hashing, never exact equality) [9, 10]. A supermajority of attested verifiers decides the verdict; an honest majority of bonded verifiers is the trust basis. Catches the one-off tamperer.
- *Future fallbacks — ZK and bisection (not in the initial implementation).* A succinct on-chain ZK proof (groth16 on bn254) [13, 14] or an interactive bisection fraud proof [11, 12] would add a trustless floor that survives even a TEE-attestation compromise. Both are deferred

(see the verification section): ZK proving is still too slow for large models, and bisection requires canonical deterministic arithmetic the initial release does not mandate.

Each layer targets a distinct cheater: TEE the lazy operator, watermark the per-output substitute, equality-testing the systematic substitute, and the attested-verifier committee the one-off tamperer.

4. Slashing & anti-collusion. Size-amplified slashing (penalty scales superlinearly with bond/throughput) makes large coordinated cheating uneconomic, and random VRF committee assignment stops verifiers from predicting or colluding on their targets. Verifier-facing incentives — verification pool, catch bounties, honeypots, minority-slashing — are detailed under Verifier incentives below.

5. On-chain vs off-chain split.

- *Consensus layer (native protocol feature):* the b_eff forging-weight rule, generating-balance slashing, and validation of finalized work balances.
- *On-chain state:* registry, bonds, verified-work balances, reward/emission, slashing, governance, Merkle-root checks of work commitments, optional groth16 verification.
- *Off-chain (node + validator software):* TEE inference, attestation-quote verification, watermark checks, Model Equality Testing audits, logit spot-checks, attested-verifier committee consensus, VRF committee assignment, epoch batching.

Verifying correctness of computation

This section gives the concrete protocol behind “proof of inference”: what a node commits to, what a verifier actually recomputes, how nondeterminism is defeated, and what touches the chain.

What a logit is

When a language model reads text, it processes it one position at a time. At each position it outputs, for every token in its vocabulary (tens of thousands of possible tokens), a raw numerical score called a logit. The whole set of logits at one position is a vector as long as the vocabulary. A softmax turns that vector of scores into a probability distribution — “how likely is each possible next token.” The model then picks the next token from that distribution (the most likely one, or a random draw weighted by it).

Two properties make logits the right thing to verify rather than the visible text:

They are deterministic given the input and the model. All the randomness in generation lives in the sampling step that happens after the logits are computed. The logits themselves are a fixed function of (model, input).

They are a fingerprint of the exact model. A different model — or the same model quantized, fine-tuned, or swapped for a cheaper one — produces measurably different logit vectors, even when the sampled text reads almost the same. This is precisely the substitution attack we must catch [4].

So verification compares logits, not generated text.

Why re-running and comparing text fails

The naive idea — “the verifier re-runs the prompt and checks it got the same answer” — does not work, for two compounding reasons. First, generation samples from the distribution, so two honest runs produce different text by design. Second, even with sampling off, one slightly different token early in the sequence cascades: the model conditions each token on all previous ones, so a single divergence at position 3 produces a completely different completion from position 4 onward. Comparing free-running outputs is therefore meaningless. The fix is to never let the verifier generate freely.

The core check: teacher-forced, single-step logit comparison

The verifier does not generate. It takes the node’s claimed output and forces the model down that exact path:

1. During the original run, the enclave commits — per output position i — to a compact sketch of that position’s logit vector, assembled into trace_commitment = MerkleRoot([H(sketch(ℓ_1)), …, H(sketch(ℓ_n))]). This root goes in the receipt; the full logits stay with the node.
2. After the epoch’s randomness beacon is revealed, a VRF picks a random position j in a sampled job (random, so a cheater can’t safely tamper only with late tokens).
3. The node reveals ℓ_j plus its Merkle path. The verifier checks the path opens against the committed root — proving the node isn’t inventing a logit after the fact.
4. The verifier takes the claimed prefix (prompt + the node’s own tokens $y_1 \ldots y_{j-1}$), runs one forward pass of model M over it — no autoregressive decoding — and obtains its own logit vector ℓ'_j.
5. It accepts if ℓ_j and ℓ'_j match within tolerance (below).

This asks the only well-posed question: given this exact prefix, is the next-token distribution consistent with model M? Sampling and cascade nondeterminism are gone because nothing is sampled and nothing autoregresses. One forward pass is cheap relative to the full generation it verifies. Verifiers can always run M because the network’s models are a finite, pinned set keyed by model_hash.

Defeating nondeterminism

Even one forward pass is not bit-identical across machines. There are three distinct sources; two are eliminated, one is bounded.

Sampling nondeterminism — eliminated by construction. Temperature, top-p, and seed only affect the pick after the logits. Teacher-forced logit comparison never samples, so this disappears. (The receipt still records decode_params and seed so the sampling step itself can be replayed if ever needed.)

Batch/sequence nondeterminism — eliminated by required kernels. On a GPU, a sequence’s logits subtly depend on what else is in the batch, because the order of floating-point reductions in attention and matrix multiplies shifts with batch size, sequence length, and padding. The same prompt in a batch of 4 vs 40 yields slightly different logits — the dominant real-world source of LLM nondeterminism. The fix is batch-invariant kernels [8]: reductions written so a sequence’s result is independent of its batch neighbors. Verifiable jobs must run on batch-invariant kernels, so the verifier (running the single sampled sequence) reproduces what the

node computed inside its large batch. Without this requirement the node and verifier are not even computing the same function.

Hardware/precision nondeterminism — bounded, not eliminated. Different GPU generation, kernel version, or numeric precision (FP16 vs BF16 vs FP8) round differently; an H100 and an A100 cannot be made bit-identical. Two measures contain it: (a) pin the environment in the attestation — the TEE quote attests not just model_hash but the kernel/runtime version and precision, and verifiers re-run in that same pinned class, narrowing the gap to same-class rounding noise; (b) tolerant matching, never equality — compare with top-k agreement, KL divergence, or locality-sensitive-hash distance under a threshold τ, the approach engineered in TOPLOC [9] and DiFR [10].

The residual research risk is calibrating τ. The attack we exploit is an empirical separation: the logit difference caused by legitimate same-class rounding is measurably smaller than the difference caused by running a different or quantized model. Set τ too tight and honest nodes are slashed on rounding drift (false positives that confiscate bonds); too loose and a tuned cheaper model slips under it (false negatives). The separation is real but its width depends on the model, the precision, and how heterogeneous the hardware fleet is — so τ must be measured on the actual target fleet, and a narrower permitted hardware/precision class widens the safety margin. Two design choices make this robust rather than knife-edge: aggregate many sampled positions across many jobs into a statistical two-sample test (a tunable p-value, not a single-token verdict), and gate slashing on a statistically significant pattern of deviation, so an honest node's occasional outlier is a non-event while a systematic substituter accumulates overwhelming evidence. Model Equality Testing [5] is the zero-sampling backstop: it compares output distributions over many samples and so sidesteps the per-logit threshold entirely.

The verdict: attested-verifier consensus

A single verifier's pass/fail is not the final word — the protocol decides each sampled job by consensus among several independent, attested verifiers, and this is the system's primary correctness mechanism.

For each sampled job, a VRF selects a committee of k verifiers. Each runs the teacher-forced logit check above inside its own attested TEE, in the pinned environment, and signs its result in-enclave.

Because every verifier is attested and runs the same pinned model, kernels, and precision, honest verifiers agree within τ. A supermajority (e.g. ⌈2k/3⌉) on fail finalizes a slash; a supermajority on pass clears the job. The node's own claim is just one more input, weighted no more than a verifier's.

If the committee splits with no supermajority either way, the protocol expands it with additional VRF-selected verifiers until a supermajority forms or a cap is reached; an unresolved job defaults to no slash (fail-safe — never confiscate a bond on an inconclusive verdict).

Verifiers are bonded. A verifier in the losing minority of a clear supermajority is slashed, making lazy rubber-stamping and minority collusion negative-EV and pushing the committee toward honest agreement.

Trust model. The correctness guarantee rests on two assumptions, stated plainly: (1) hardware-vendor attestation (Intel/NVIDIA) is sound [2, 3], and (2) an honest majority of bonded verifiers holds. This is a real, bounded trust assumption — not trustlessness. The one threat it does not

cover is a correlated compromise of the attestation root itself (e.g. a leaked vendor key), which would let many verifiers attest the same wrong result; redundancy cannot defeat a fault that is correlated across all the redundant parties. That residual risk is what the deferred fallbacks below are designed to close.

Future fallbacks (not in the initial implementation)

Two mechanisms would add a trustless floor that survives even a compromised attestation root. Both are deliberately out of scope for the first release and recorded here as the upgrade path:

Single-shot ZK proof on dispute. On a contested slash, the node produces a succinct proof (groth16 [14] on bn254, verified cheaply on-chain) that the forward pass was computed correctly. Sound and needs no determinism-matching, because the circuit defines the arithmetic. Deferred because proving a forward pass is still too slow and expensive for large models — zkLLM reports on the order of hundreds of seconds for a ~13B pass [13], worse above — feasible first for smaller models.

Interactive bisection fraud proof. An Arbitrum-style binary search [12] that narrows a disputed computation to one primitive operation the chain executes itself, as in opML [11]. Deferred because it requires canonical, bit-exact arithmetic — which GPU floating point is not: honest machines legitimately differ, and τ-tolerant comparison has no clean “first divergence” to search for. It becomes viable only if verifiable inference runs in fully-specified integer/fixed-point arithmetic, a larger change with its own quality and throughput trade-offs.

Until those land, the initial release relies on attested-verifier consensus plus the statistical layers (Model Equality Testing, watermarking), under the trust model stated above.

What gets tracked — and what touches the chain

Individual inference calls are never posted on chain. Millions of jobs, private prompts, and per-job gas make that impossible. Data lives in three tiers:

Tier	Contents	Where
On chain (permanent)	one Merkle root of receipts per node per epoch, total_work_units, num_jobs, a data-availability pointer; plus verdicts, disputes, slashings	consensus state
Available (on demand)	the individual signed receipts (hashes, TEE quote, trace commitment)	node-served / DA layer
Private (revealed only if sampled)	the actual prompt, output tokens, full logit trace	node + requester

The mandatory on-chain footprint is O(1) per node per epoch, independent of how many jobs the node served.

The per-job receipt

Each call produces one small (~1–2 KB) receipt, held off chain:

```
receipt = { H(request), H(output), model_hash, kernel_version, precision,
            decode_params, seed, work_units, timestamp,
            trace_commitment,   // Merkle root over per-position logit sketches
            tee_quote,          // enclave-signed over the hashes, model_hash,
work_units
            node_sig, payer_sig }
```

Two signatures carry distinct meaning. The TEE quote attests correctness (real model, pinned environment, executed in-enclave). The payer co-signature attests existence — the job is corroborated by the wallet that paid for it, not just the node's claim. This is what ties verification to the self-dealing defense: because the forging boost counts only jobs paid by a distinct, staked payer who co-signs, fabricating boost-eligible work requires a colluding staked payer, not a node acting alone.

Lifecycle of a verified job

1. Execute (off chain). Requester sends a signed request (prompt may be encrypted to the enclave). The node runs it in the TEE and returns (output, receipt); the requester counter-signs to accept. Both hold the receipt.
2. Commit (on chain, once per epoch). The node Merkle-trees the epoch's receipts and posts {node_id, epoch, receipts_root, total_work_units, num_jobs, DA_pointer}. This is the entire required on-chain write.
3. Sample (post-beacon). The revealed randomness beacon seeds a VRF that selects a random subset of jobs and positions. The node must open those leaves and produce the full payloads; failure to produce is an automatic slash, so withholding is self-defeating.
4. Verify by committee (off-chain compute, on-chain verdict). A VRF-selected committee of attested verifiers independently re-checks each sampled job and signs results in-enclave. A supermajority verdict is posted as {node_id, epoch, job_ids, pass/fail, verifier_sigs}. Pass is silent; a supermajority fail slashes the node, and any verifier in the losing minority is slashed too.
5. Resolve splits (only if inconclusive). If no supermajority forms, the committee is expanded with more VRF-selected verifiers until it does, or — at the cap — the job defaults to no slash. (The deferred ZK/bisection fallbacks would attach here as a trustless tier.)
6. Finalize. After a challenge window with no successful fraud proof, total_work_units finalizes into the canonical verified-work balance and feeds workBoost for forging in epoch E+1. The E→E+1 lag in the consensus design is this challenge window.

Why checking a few percent is enough

We deliberately verify a random fraction p of jobs, not all of them, and pair sampling with whole-bond slashing. A node faking a fraction φ of its jobs evades detection in an epoch with probability about $(1 - p)^{(\varphi \cdot m)}$ for m jobs — so even p ≈ 1–5% catches systematic cheating with near-certainty once m is large, and losing the entire bond on a single catch makes cheating negative-EV even for small φ. The honest limitation: you cannot economically catch a node faking one job in a million — but such a job earns negligible fees and moves workBoost immeasurably, so it does not matter. Only systematic cheating is profitable, and that is exactly what sampling plus slashing, with Model Equality Testing underneath, is built to catch.

The one real tension: private inputs

The logit spot-check needs the verifier to see the prefix and re-run it. For public/auditable jobs this is fine. For jobs whose prompt is encrypted to the enclave, the plaintext cannot be handed to an arbitrary verifier. Three options, used in tiers:

Private jobs lean on TEE attestation plus output-only statistical checks (Model Equality Testing, watermarking) that need no plaintext — losing per-job logit checking but keeping substitution detection.

Requester-consented audit: the payer agrees up front that sampled jobs may have their input revealed for verification — acceptable because only a tiny fraction is ever sampled.

Validator-in-TEE: the spot-check runs inside the validator's own enclave, which decrypts and re-runs without exposing the plaintext — privacy-preserving, at the cost of adding the validator's TEE to the trust surface.

A typical deployment defaults to the first for private traffic, the second for public traffic, and reserves the third for high-value private jobs.

Verifier incentives

Verification must pay in the honest case, or verifiers stop showing up exactly when the network is healthy — and security quietly evaporates. Four mechanisms, layered:

Verification fee (the base salary). Every job's payment is split: most to the serving node, a slice (≈5–10%) into a per-epoch verification pool. The committee assigned that epoch splits the pool for performing the checks, whether or not anything is found. Honest-case income that scales with network volume.

Verification is attested work that earns forging weight. Re-running a sampled job is itself real, checkable inference, so it feeds the verifier's workBoost exactly as serving does. This folds verifiers into the consensus incentive already in place — a node alternates between serving and verifying, both raise b_eff, and neither pays better by construction. (This does not contradict the rule that a node cannot earn weight by replaying its own cached work: verifying another node's VRF-assigned job is distinct, unpredictable, non-self-dealable work.)

Catch bounty (the upside). A verifier in the supermajority that slashes a cheating node earns a share of the forfeited rewards. This rewards thorough rather than lazy checking and funds the cost of pursuing a contested verdict.

Honeypots (the stick). Known-bad jobs are salted into the verification stream; a verifier that passes one is slashed. Combined with minority-slashing in the consensus step, this makes rubber-stamping negative-EV, so the base salary pays out only if the verifier actually computes.

Salary plus forging-weight give verifiers a reason to show up; honeypot risk, minority-slashing, and catch bounties give them a reason to actually compute and to converge on the honest answer.

Security invariants

- Work multiplies stake, capped; never replaces it. Security floor = pure PoS; it only rises with useful work. No bond → no boost; no stake → no forging, regardless of compute.
- Only pre-finalized, verified, real-demand work boosts forging. Boost reads epoch E−1 finalized state; never live, unverified, or self-issued work.
- Self-dealing the boost is made unprofitable, not detected: boost value capped below compute cost ($\beta \le c$), 20–30% fee burn, and boost gated on an external staked payer. Per-device caps and emission-per-job < fee-per-job still apply.
- TokenZ claims stay capacity-backed — staker emission is bounded by realized servable inference, never minted faster than the network can deliver.
- Self-replay never earns; assigned verification does. A node cannot gain weight by replaying its own cached work; verifying another node's VRF-assigned job is distinct attested work and is rewarded (see Verifier incentives).
- VRF hit untouched — grinding resistance preserved.
- Fixed decoding parameters for verifiable jobs — declared params + committed seed, or no layer has a clean reference to compare against.
- Defense-in-depth verification — a cheater must defeat hardware attestation and evade independent statistical checks (equality-testing, watermark) and an honest-majority committee of attested verifiers re-executing the work.
- Slashing verdicts must be defensible — prefer statistical thresholds (p-values) and deterministic re-execution over subjective judgments, since verdicts suspend a node's generation.

Net effect

Useful inference strengthens consensus when demand is present and disappears cleanly when it isn't, with security always anchored in WAVES stake, on mainnet with the real token. Costs and caveats: it is a consensus-breaking change that must pass Waves' feature-activation voting (network governance, not just a merge); the non-slashable LPoS model is preserved by penalizing proven cheating through generation-suspension rather than principal confiscation; WAVES value-capture from inference must be wired explicitly; and there is a bounded centralization pressure (most-compute earns extra forging weight on top of most-stake), controlled by B_{max}. The verification pipeline must always run ahead of consensus use, since work that buys block-production probability must be fully verified before it counts.

Appendix: economic security (worked model)

Parameters: WAVES $0.20; 20% of ~120M supply staked; 4 WAVES/block at ~60s reward-block interval; $B_{max} = 2$; marginal compute cost $c \approx \$0.0004$ per 1k-token job.

Derived: reward pool = $4 \times 1440 \times 365 \approx 2.10$M WAVES/yr ≈ $420k/yr (independent of supply); staked = 24M WAVES ≈ $4.8M; base staking yield ≈ 8.8%. The entire forging pie is ~$420k/yr — an upper bound on what the boost can be worth to anyone.

Value of a maxed boost (adds $\approx s \cdot B_{max} \cdot R$, where s = stake share):

Staker	Stake	s	Base reward/yr	Max-boost adds/yr
Small	10k WAVES ($2k)	0.04%	~$175	~$350
Mid	120k WAVES ($24k)	0.5%	~$2,100	~$4,200
Whale	1.2M WAVES ($240k)	5%	~$21,000	~$42,000

A maxed node triples forging rewards (~8.8% → ~26% yield) where uncontested; under full participation the rent is competed away into real capacity (“serve or be diluted”).

Self-dealing break-even. Self-dealing nets s·B_max·R − W*·c, unprofitable when W* > s·B_max·R / c. The whale is the marginal attacker (boost value scales with stake; compute cost does not). At these parameters: W* > 0.05 × 2 × $420,480 / $0.0004 ≈ 105M jobs/yr ≈ 2.5 GPU-years. So a 5%-staker must run ~2.5 H100s flat-out (~$44k compute) to capture ~$42k of boost — break-even at the whale, a loss for everyone smaller, before any burn. Natural calibration: set “work to max boost” ≈ one-to-a-few GPU-years of output.

Burn is margin, not deterrence, here. Since the boost is cheap relative to compute at $0.20, β ≤ c holds almost for free, so the burn required purely to deter is ≈ 0. The 20–30% fee burn is value-capture and safety margin.

Price scaling (the invariant to encode). Deterrence scales with the reward pool, hence with WAVES price. At $2.00 (10×), the pool → $4.2M/yr and the whale break-even rises to ~25 GPU-years; self-dealing turns profitable above ~0.5% stake unless W *rises 10×. Therefore “work to max boost” must scale with WAVES_price × B_max / c — a governance-tracked value pegged to on-chain price, not a fixed constant. As WAVES appreciates or inference cost falls, raise W* (or cut B_max) in proportion.

References

1. Waves Protocol. *Leased Proof-of-Stake and Fair PoS consensus.* Waves documentation, docs.waves.tech.
2. Intel Corporation. *Intel Trust Domain Extensions (Intel TDX).* Whitepaper, 2023.
3. NVIDIA. *NVIDIA Confidential Computing (H100 Tensor Core GPUs).* Technical documentation, 2023.
4. W. Cai, T. Shi, X. Zhao, D. Song. *Are You Getting What You Pay For? Auditing Model Substitution in LLM APIs.* arXiv:2504.04715, 2025.
5. I. Gao, P. Liang, C. Guestrin. *Model Equality Testing: Which Model Is This API Serving?* arXiv: 2410.20247, 2024 (ICLR 2025).
6. A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, A. Smola. *A Kernel Two-Sample Test.* Journal of Machine Learning Research 13:723–773, 2012.
7. J. Kirchenbauer, J. Geiping, Y. Wen, J. Katz, I. Miers, T. Goldstein. *A Watermark for Large Language Models.* ICML 2023; arXiv:2301.10226.
8. H. He and Thinking Machines Lab. *Defeating Nondeterminism in LLM Inference.* Thinking Machines Lab: Connectionism, 2025. doi:10.64434/tml.20250910.
9. J. M. Ong, M. Di Ferrante, A. Pazdera, R. Garner, S. Jaghouar, M. Basra, M. Ryabinin, J. Hagemann. *TOPLOC: A Locality Sensitive Hashing Scheme for Trustless Verifiable Inference.* arXiv:2501.16007, 2025.

10. A. Karvonen, D. Reuter, R. Rinberg, et al. *DiFR: Inference Verification Despite Nondeterminism.* arXiv:2511.20621, 2025.
11. K. D. Conway, C. So, X. Yu, K. Wong. *opML: Optimistic Machine Learning on Blockchain.* arXiv:2401.17555, 2024.
12. H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, E. W. Felten. *Arbitrum: Scalable, Private Smart Contracts.* USENIX Security Symposium, 2018.
13. H. Sun, J. Li, H. Zhang. *zkLLM: Zero Knowledge Proofs for Large Language Models.* ACM CCS 2024; arXiv:2404.16109.
14. J. Groth. *On the Size of Pairing-Based Non-interactive Arguments.* EUROCRYPT 2016.