THOX.ai Research · Technical report v1.1.1 · 2026-11-10 Preprint, not peer reviewed

THOX Community Mesh: Unlinkable Subscriber Tokens, Digest-Audited Stage Execution and a Witness-Cosigned Settlement Ledger for Member-Contributed Inference

How MeshStack™ subscribers can pool their own hardware, and reach cloud models through it, without the pool learning who is asking: a design, a working virtual lab and the measurements behind both.

Craig Ross and Tommy Xaypanya · Corresponding author: Tommy Xaypanya, CTO and Co-Founder, THOX.ai LLC · ORCID 0009-0000-4425-9565

ABSTRACT

Most people who want to run a capable language model can afford neither the hardware nor a metered cloud account, while their neighbours' computers sit idle. This paper specifies and tests a community mesh for THOX MeshStack™: members add their own devices to a shared pool, and any subscriber can run distributed inference on that pool, or reach a hosted cloud model through it, without the pool learning who is asking. The design combines blind-signed subscriber tokens (RFC 9474) and sealed envelopes through a relay (RFC 9180) for requester unlinkability; signed per-hop receipts with sampled replay audits for integrity; contribution credits that raise a member's token budget and tier, with no currency; and a witness-cosigned Merkle ledger that settles credits and penalties after the request, off the token path. We state the limit plainly: a contributor that executes a stage can read its activations, so the pool gives identity privacy, not content privacy, and requests that need the latter are refused rather than routed. On a real 135M-parameter model split into three stages, a token-exact re-execution audit, the natural choice, exposed 69 of 102 cheated requests and missed an fp16 weight substitution on all 17 prompts; a SHA-256 digest of the stage boundary exposed 102 of 102. Minting an unlinkable token costs 7.9 ms at RSA-2048 and verifying one 0.12 ms; the ledger costs 0.92 microseconds per receipt. A virtual lab of 15 processes with real cryptography ran 12 scenarios, including a contributor killed mid-request, an owner reclaiming a device, a cheating contributor caught and penalised, and a forged ledger head refused by every witness; all passed and every lab request was token-identical to the whole model. A simulation shows the cost of unlinkability for fairness: fixed issuance budgets isolate heavy users but waste capacity, and indexing issuance to pool load recovers the waste. This edition is consolidated with a companion THOX paper and lab on the same subject: we adopt its plan-wide admission rule, execution contexts and replica-first default, reproduce its 62 tests and 150-trial campaign on a second host, and measure the replica-first claim in our own lab (6.6% lower per-remote-call round trip for one contributor holding both remote ranges than for two contributors in sequence on loopback, 30 matched pairs; corrected in v1.1.1, see Section 6.7). Nothing here is a THOX device, wide-area or throughput result.

CONTENTS

Executive summary
1. Introduction
2. Background and related work
3. System and threat model
4. Design
5. Implementation

Executive summary

MeshStack™ lets one owner pool their own devices. This report specifies the next step: a community pool that different owners contribute to and any subscriber can use, plus a path to hosted cloud models through the same private front door. It was written after a review of the five MeshStack repositories on 2026-11-10 and it reuses what they already decide: the lease and owner-protection contract of Auto Device Role Mode [1], exact-identifier model sharing, fail-closed admission, and the decisions of record [2]. Where the brief for this paper met an existing decision, the design stays inside the decision and says so. This edition (v1.1.1, erratum version of the v1.1.0 edition) is consolidated with the companion paper *MeshStack Community Compute* and its virtual lab [3], prepared in parallel on the same day. Section 6.8 reproduces that lab on this host and records where the two works agree, where they differ and who decides.

102 / 102
cheated requests exposed by a boundary-digest audit; a token-exact audit exposed 69 and missed fp16 weights on all 17 prompts (SmolLM2-135M-Instruct, six cheat classes)
MEASURED

7.9 ms
to mint one unlinkable subscriber token at RSA-2048 (client 0.65 ms, issuer 7.25 ms); 0.12 ms to verify; RFC 9474 test vector reproduced
MEASURED

0.92 µs
settlement ledger cost per receipt at 16,384 receipts per epoch; inclusion proof 448 bytes; nothing added between tokens
MEASURED

12 / 12
virtual lab scenarios passed across 15 processes; 8 of 8 community-plane requests token-identical to the whole model
VIRTUAL LAB

PRODUCT IMPLICATIONS

First, the community plane can promise that the pool does not learn who is asking. It cannot promise that the contributor running a stage does not learn what was asked, and the product must say so at the point of use; personal work stays `local_only` by default. Second, contributor receipts must carry boundary digests. Token-exactness remains the right gate for certifying a stage decomposition (decision D25) and the wrong predicate for auditing a stranger. Third, contribution can earn access without a coin: non-transferable credits raise a member's token budget and tier, which keeps decision D20 intact. Fourth, hiding the requester moves fairness from the scheduler to token issuance, and a fixed issuance budget is expensive: in simulation it cut served tokens by 28% at 0.6 load. Indexing issuance to one aggregate idle bit from the pool recovers that loss in simulation and works in the lab, but does not isolate heavy members once the pool is saturated. Fifth, prefer a whole remote replica to a partition wherever one device can hold the remote part of the model: fewer hops, fewer parties that see activations, one receipt per token. Sixth, a blockchain-like ledger belongs after the request, not inside it: a witness-cosigned log gives append-only, publicly checkable settlement for microseconds per receipt.

1 Introduction

ERRATUM IN V1.1.1

This version corrects five points in v1.1.0, found by independent review after publication. (1) Proposition 4 now states its assumption: the no-quiet-rewrite property requires that no witness is faulty, and the witness rule checks head chaining, not log-content prefix containment (Section 4.5). (2) The route comparison in Sections 1 and 6.7 used different prompts on each route; it is replaced by a matched-prompt rerun (6.6% in place of 7.5%). (3) The route called a whole replica holds the remote layers only; the requesting device retains the first ten layers in both routes (Sections 3 and 6.7). (4) The fairness waiting times cover requests that started; the never-started accounting is added (Section 6.4). (5) The limitations now state that the lab control services are not a security boundary and list the defects found in them (Section 7). No other result changes.

Running a capable language model costs either a large computer or a metered cloud account. Most households and small teams have neither, yet between them they own a great deal of idle memory and compute. MeshStack™ already lets one owner pool their own devices: the previous THOX report specified pipeline sharding, a keyed KV graph and lease-fenced device roles for a single household [4], and Auto Device Role Mode lets an owner's device lend itself to that owner's mesh under a lease [1]. This paper takes the next step and asks what has to be true before *different* owners can pool hardware: a member adds a device to a community pool, and any MeshStack subscriber can run inference on that pool, or reach a hosted cloud model through it, without the pool learning who they are.

Crossing the ownership boundary changes the problem. Inside one household every device answers to the same person. In a community pool the device that executes a stage belongs to a stranger, the requester is a stranger to that device, and neither has a reason to trust the other. Volunteer computing met the same boundary two decades ago and answered with redundancy and spot checks [5, 6]. Petals showed that a large language model can be served by volunteers holding layer ranges [7, 8]. What neither settles for a consumer product is the combination THOX needs: the owner of a contributed device stays sovereign, the requester stays unidentifiable, cheating is caught and costs more than it earns, contribution earns access without a currency, and none of the bookkeeping slows a token.

We are direct about one limit from the first page. A device that computes on a request can read what it computes. Recent work reconstructs prompts from the intermediate activations that pipeline stages exchange [9], and the THOX security model already records the same fact for KV state [10]. The community plane in this paper therefore offers *identity* privacy, not *content* privacy against the executing contributor. Requests that need content privacy stay on the requester's own devices, and the system refuses to route them anywhere else.

Contributions:

1. **A design for a member-contributed pool on MeshStack** with four planes (local, own mesh, community, cloud), ten numbered requirements, and each requirement mapped to a test that ran (Sections 3 to 5).
2. **Requester unlinkability carried by blind-signed tokens** (RFC 9474) and sealed envelopes through a relay (RFC 9180), with priority carried by the token's issuing key so that contribution can raise priority without identifying the requester (Section 4.2).
3. **A measured comparison of two audit predicates on a real model.** Re-executing a request and comparing output tokens, the gate MeshStack already uses to certify stage decomposition, misses a contributor running fp16 weights on 17 of 17 prompts. Comparing a SHA-256 digest of the stage boundary catches every cheat class we tried on every prompt (Section 6.2).
4. **A settlement ledger that behaves like a blockchain where it matters and costs nothing where it does not**: signed receipts in band, Merkle batching and witness cosigning after the request, measured (Sections 4.5 and 6.5).
5. **A working virtual lab**: 15 operating-system processes with real cryptography and a real model, 12 scenarios, one evidence file. Every lab claim in this paper is a field of that file (Section 6.6).
6. **Consolidation with the companion work** [3]: its plan-wide admission rule, execution contexts and replica-first default are adopted (Section 4); its lab is reproduced on a second host and its decode-cycle model is re-derived and applied to this design (Sections 6.7 and 6.8).

2 Background and related work

2.1 Pooled inference across owners

Petals serves large models by letting volunteers host contiguous transformer blocks and chaining them per request, with fault-tolerant routing when a server disconnects [7, 8]. Parallax schedules layer placement and request-time pipeline selection over a decentralised pool of heterogeneous GPUs [11]. Prima.cpp targets home clusters of mixed, low-resource devices [12]. These systems establish that the pipeline form works over consumer links; they are the base this paper builds on, not a contribution of it. None of them hides the requester from the pool, and none binds admission to a subscription.

2.2 Verifying work done by strangers

Volunteer computing verifies results by replication and spot checks; Sarmenta's analysis of spot-checking with blacklisting and credibility is the classical treatment [6], and BOINC ships replication-based validation [5]. For model inference, optimistic schemes re-execute on dispute [13], Proof of Sampling derives an equilibrium in which sampled verification makes honesty the rational strategy [14], Proof of Quality replaces exact verification with quality consensus [15], SVIP trains a proxy task on returned hidden states to detect model substitution [16], and TOPLOC commits to intermediate activations with a locality-sensitive hash that tolerates hardware differences [17]. Our audit is the simplest member of this family: an exact digest of the stage boundary, replayed by an auditor in the same determinism class. Its only novelty is the measurement in Section 6.2 that the weaker, more natural predicate (compare the tokens) does not work, and the binding of the audit to signed receipts and a public ledger.

2.3 Asking without being identified

Privacy Pass separates the party that attests a client from the party that serves it, using tokens the issuer cannot link to their redemption [18, 19]; the publicly verifiable variant uses RSA blind signatures [20]. Oblivious HTTP places a relay between client and gateway so that the relay sees the client's address but not the request and the gateway sees the request but not the address [21], building on HPKE [22]. Apple's Private Cloud Compute combines both, an OHTTP relay and RSA blind-signature credentials, in front of attested servers [23]. Onion routing generalises the relay to multiple hops [24]. We adopt the Privacy Pass and OHTTP split unchanged and apply it to a pool whose servers are other members' devices.

2.4 What a stage can learn

Embedding inversion recovers text from dense embeddings [25] and from next-token distributions [26]. For the pipeline setting specifically, prompt inversion

against collaborative LLM inference reconstructs inputs from intermediate activations [9], and proposed defences trade utility for reduced leakage without eliminating it [27]. Cryptographic private inference remains orders of magnitude slower than plaintext [28]. Trusted execution environments bound the exposure with a modest overhead on server CPUs [29] and are the long-term answer the THOX security model already names [10]; contributed consumer devices do not offer one today.

2.5 Sharing fairly

The Virtual Token Counter serves the least-served client first and lifts a returning client's counter so idle time banks no credit [30]; later work adds prefix locality [31] and per-token latency isolation [32]. Dominant Resource Fairness is the multi-resource baseline [33]. All of these keep a counter per client. Section 4.4 explains why an unlinkable pool cannot, and what replaces it.

System	Hardware from	Requester hidden from pool	Result verification	Access model	Owner pre-emption
Petals [7, 8]	volunteers	no	none stated in the abstracts we cite	open	server may leave
Parallax [11]	decentralised GPU pool	no	not in scope	open pool	dynamic availability
BOINC [5]	volunteers	n/a (project is requester)	replication	project-run	volunteer preferences
Proof of Sampling [14]	staked nodes	no	sampled re-execution, game-theoretic	payment	n/a
Private Cloud Compute [23]	one operator	yes (relay and blind tokens)	attested software	device entitlement	n/a
This paper	subscribers' own devices	yes (relay and blind tokens)	boundary-digest audit, signed receipts, public ledger	subscription, non-transferable credits	owner always wins within one chunk

Table 1. Positioning. Entries for other systems restate their own published descriptions; they are not measurements by THOX.

3 System and threat model

3.1 Parties

- **Member.** A MeshStack subscriber. A member may be a *requester*, a *contributor* (owner of one or more enrolled devices), or both.
- **Subscription authority.** Knows accounts. Issues device certificates and blind-signed tokens. Holds the credit balance that sets issuance budgets.
- **Coordinator.** Keeps pool membership, redeems tokens, issues leases, accepts fraud proofs. Never learns an account.
- **Relay.** Forwards sealed envelopes. Holds no decryption key.
- **Stage node.** A contributed device holding a contiguous layer range of one certified model.
- **Auditor.** A node in the same determinism class as the stage it checks. It replays sampled hops.
- **Cloud gateway.** A hosted model provider reached through the relay. It can read the prompt; it cannot tell whose it is.
- **Ledger and witnesses.** An append-only log of credits and penalties, and independent parties that cosign its heads.

3.2 Adversaries

Curious contributor: follows the protocol and tries to learn who is asking and what. **Lazy or dishonest contributor:** returns cheaper or wrong results to earn credit (skips layers, substitutes lower precision, perturbs outputs). **Free rider:** a subscriber, or a non-subscriber, who tries to consume more than their share. **Curious operator:** the coordinator or the authority tries to link requests to accounts, or rewrites the credit history. **Network observer:** sees traffic between parties. Out of scope: a contributor reading the activations it legitimately computes (Section 1), collusion between the relay and the coordinator, traffic-timing correlation, and physical attacks on devices.

3.3 Requirements

	Requirement
R1	**Owner sovereignty.** Contribution is opt-in per device and per exact model identifier. An owner who returns to their device waits at most one service chunk; community work never evicts owner work.
R2	**Requester unlinkability.** No party other than the subscription authority learns which member made a request, and the authority cannot link a token it issued to the request that redeems it.

	Requirement
R3	**Stated content exposure.** Every request carries a privacy class. `local_only` never leaves the device; `mesh_only` never leaves the owner's devices; `community` accepts that the executing contributor can read activations; `cloud` accepts that the provider can read the prompt; `confidential` requires an attested execution boundary on every stage that handles plaintext, which no contributed consumer device offers today, so it is refused. A request is refused, not downgraded, when its class forbids a plane.
R4	**Subscriber-only, bounded admission.** Only tokens issued to subscribers are accepted. Each token authorises one bounded service quantum and is spent once. Issuance per account per epoch is bounded.
R5	**Auditable integrity.** Every hop yields a signed receipt binding input and output digests. A contributor that returns a wrong result is caught with probability approaching one under sampled audit, and loses more than it gained.
R6	**Contribution earns access without identifying the requester.**
R7	**Plane provenance.** Every response is labelled with the plane that produced it. A cloud-hosted model is never advertised as contributed compute [34].
R8	**No cross-member cache.** KV state stays keyed to and readable by one principal, as decisions D19 and D43 require [2].
R9	**Verifiable settlement off the token path.** Credits and penalties are append-only, publicly checkable and replayable, carry no requester identity, and add no work between two generated tokens.
R10	**Fail closed with attribution.** A failed hop, an audit mismatch, a broken receipt chain or a stale lease stops that path and names the hop. Recovery is a new route, never a silent degradation.

Table 2. Requirements. Section 5 maps each to the test that exercised it.

4 Design

4.1 Planes and seat order

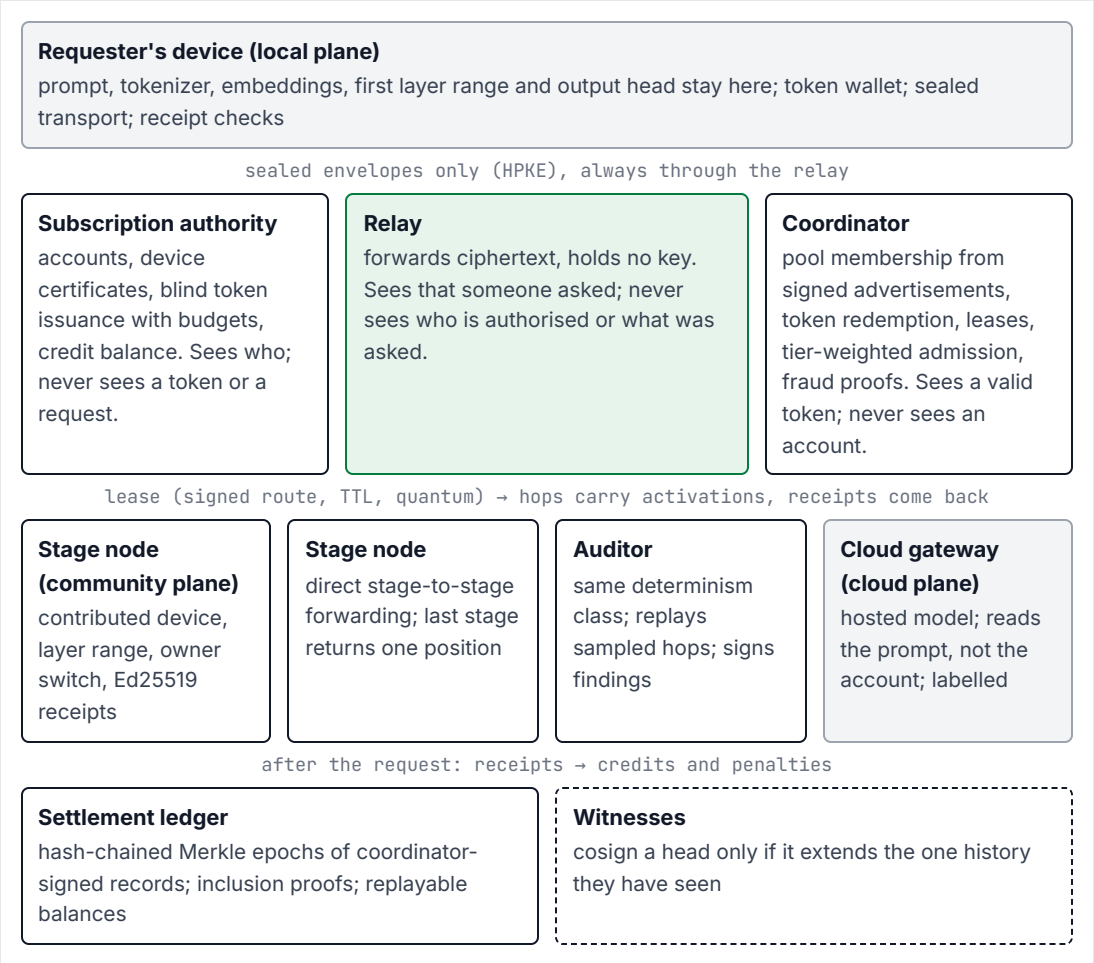


Figure 1. Parties and what each can see. Shaded boxes are not operated by members. The own-mesh plane of the previous report sits between local and community and is unchanged.

A request tries planes in a fixed order and stops at the first that both fits the model and is permitted by the privacy class: the local model, the owner's own mesh [4, 1], the community pool, then the cloud plane. This extends the seat order of Auto Device Role Mode by one seat and keeps its rule that an empty pool is a fall-through, not an error. The plane that served a response is part of the response.

The privacy classes of R3 correspond to the execution contexts of the companion paper [3], which we adopt: owner-controlled mesh (`local_only`, `mesh_only`), trusted community hosts (`community`), attested confidential hosts (`confidential`) and external model service (`cloud`). They are policy contexts, not a ranking. We also adopt its plan-wide admission rule. A job j may run on a plan π at time t only if

$$\mathrm{Eligible}(j,\pi,t) = E_j(t) \ \wedge\ C_j(\pi,t) \ \wedge\ R_j(\pi) \\ \wedge \bigwedge_{v\in V(\pi)} A(j,v,t) \ \wedge \bigwedge_{e\in F(\pi)} D(j,e,t)$$

where E is subscriber entitlement (here, a valid unspent token), C consent (the privacy class), R model rights and compatibility (the certified fingerprint), A admission at every stage (enrolled, owner idle, not suspended, lease valid) and D authorisation of every data edge (sealed to the next hop's key). Only eligible plans are ranked; an ineligible candidate is excluded, never given a worse price. In the lab the signed lease is the plan: it carries a plan digest over route, keys, model fingerprint, quantum and expiry, and every receipt repeats that digest.

Replica first. Also following [3], the default route is a whole replica of the remote layers (the requesting device retains the first ten layers locally in both routes): if one eligible device can hold all of the remote layers, use it, and split across devices only when no single device can and a measured benefit exists. A replica route has fewer hops, exposes activations to one contributor instead of two, and needs one receipt per token. Section 6.7 measures the difference in the lab and calculates it for wide-area links.

4.2 Tokens: authorised, bounded, unlinkable

The authority holds one RSA key per *tier*. To obtain a token the member's device draws a random 32-byte nonce, appends the intended audience (`community` or `cloud`), blinds the encoded message under the tier's public key and sends the blinded value with its account credential. The authority checks the account and the tier budget, signs the blinded value, and returns it. The device removes the blinding factor and holds a standard RSASSA-PSS signature on a message the authority has never seen.

$$z = \mathrm{EMSA\text{-}PSS}(m)\cdot r^{e} \bmod n, \qquad \hat{s} = z^{d} \bmod n, \\ s = \hat{s}\cdot r^{-1} \bmod n, \qquad s^{e} \equiv \mathrm{EMSA\text{-}PSS}(m) \pmod n$$

The redeemer (coordinator or cloud gateway) verifies s under the tier key, checks the audience, and records $\mathrm{SHA\text{-}256}(m)$ in a spent set. One token buys one request of at most Q generated tokens. The tier is visible to the redeemer because it is the key that verifies; nothing else about the holder is.

PROPOSITION 1 (UNLINKABILITY, INHERITED)

For a fixed public key and any two messages, the blinded values the authority sees are identically distributed, because r is uniform over the units modulo n and so is r^e. The authority's view at issuance is therefore independent of the token later redeemed. This is the blindness property of RFC 9474 [20]; we rely on it and do not re-prove it. The anonymity set of a redeemed token is every member issued a token of that tier in that key epoch, so tiers must stay few and keys must rotate on a schedule, not per member.

4.3 The request path

Algorithm 1. One request on the community plane

```
1  device: token ← wallet(tier, aud=community)           # issued earlier,
unlinkable
2  device → relay → coordinator: seal(lease request, token, privacy class)
3  coordinator: verify token, check spent set, refuse local_only / mesh_only,
               pick one eligible node per remote stage (not owner-active, not
suspended, not busy),
               return signed lease {route, TTL, quantum Q}
4  for step t = 0 .. Q-1:
5     device: h0 ← local layers(embed(prompt + generated))
6     device → relay → stage 1: seal(lease, t, h0)         # stage checks lease
signature, route, owner switch
7     stage 1: h1 ← layers(h0); receipt1 = sign(lease, t, SHA256(h0), SHA256(h1))
8     stage 1 → stage 2 directly: seal(lease, t, h1, receipt1)
9     stage 2: h2 ← layers(h1)[last position]; receipt2 = sign(lease, t, SHA256(h1),
SHA256(h2))
10    reply to device, sealed to its one-time key: h2, receipt1, receipt2
11    device: verify both signatures and the chain  in1 = SHA256(h0), in2 = out1,
out2 = SHA256(h2)
12    device: with probability q send (h0, receipts) to an auditor; on mismatch
submit fraud proof, reroute, restart
13    device: next token ← argmax(head(h2))                # head stays local
14 device → coordinator: close(lease, receipts)            # credits go to the
ledger after the fact
   any failure at 6-11: reroute excluding the named hop and repeat step t (hops are
stateless in the lab)
```

Three properties of this path matter. Token identifiers never leave the device: embeddings and the output head are local, so contributors receive activations only. The receipt chain lets the device detect, without any replay, a stage that signs one output and forwards another, because the next stage signs what it actually received. And the owner switch is checked on every hop, so a returning owner is served at the next hop boundary (R1).

4.4 Admission when the pool cannot count per member

Fair serving normally keeps a service counter per client [30]. A pool that cannot link two requests from the same member cannot. We move the per-member accounting to the one party that knows members, the authority, and express it as an issuance budget: each account may draw a bounded number of tokens per epoch. Contribution raises the budget and unlocks a higher tier. The coordinator

then needs only per-tier state. It grants the queued tier with the least service per unit weight, with the counter lift of VTC applied per tier so an idle tier banks nothing.

$$\text{grant tier } \tau^{*} = \arg\min_{\tau \in \text{queued}} \frac{\sigma_\tau}{w_\tau}, \qquad \sigma_\tau \leftarrow \sigma_\tau + Q \text{ on grant},$$
$$B_m = B_0 + \lfloor \beta \, c_m \rfloor$$

Here σ_τ is service granted to tier τ, w_τ its weight, B_m member m's budget and c_m the verified service that member's devices gave to others. Credits are non-transferable and buy only tokens. Decision D20 rejected a payments market for MeshStack [2]; this design stays inside that decision. The cost of moving fairness to issuance is that a budget is not work-conserving: capacity can sit idle while a member waits for their bucket to refill. Section 6.4 measures that cost in simulation. The remedy is to index issuance to pool load: the coordinator publishes one aggregate bit (a route is free and nothing is queued), and while it is set the authority issues one base-tier token past the budget. The bit identifies nobody. Contributor-tier tokens are never issued as surplus.

4.5 Audits, penalties and the settlement ledger

An audit replays a hop. The device sends the auditor the stage input it produced and the receipts it received. The auditor, holding the same layer ranges at the same engine build, recomputes each remote stage in turn and compares the SHA-256 digest of its output with the digest the contributor signed. A mismatch yields a *fraud proof*: the contributor's signed receipt and the auditor's signed finding over the same input digest with a different output digest. The coordinator verifies both signatures and suspends the node; the authority zeroes the owner's credit; the device reroutes and restarts the request so that nothing the cheater touched is kept.

> PROPOSITION 2 (DETECTION)
> Let a contributor cheat on every request, let each request be audited independently with probability q, and let an audit expose the cheat with probability d. The probability it is caught within n requests is $1 - (1 - qd)^n$ and the expected number of requests before detection is $1/(qd)$. *Proof.* Audits are independent Bernoulli trials with success probability qd; the waiting time is geometric.

> PROPOSITION 3 (DETERRENCE)
> Let g be the credit a contributor gains per cheated request and L what it loses on detection (forfeited balance and the value of suspension). Cheating has negative expected value when $q\,d\,L > (1 - qd)\,g$, that is when $q > \dfrac{g}{d\,(L + g)}$. *Proof sketch.* Per request the cheater gains g with probability $1 - qd$ and loses L with probability qd. The bound is only as good as d: an audit predicate that rarely exposes a cheat makes every audit rate insufficient, which is why Section 6.2 measures d per cheat class.

Settlement is where a blockchain-like structure earns its place, and where it must not cost latency. Receipts are signed in band because audits need them anyway. Everything else happens after the request. The coordinator signs a record per lease (member credited, token-steps served) or per penalty and appends it to a log. Each epoch the log is rolled into a Merkle tree with the hashing of Certificate Transparency [35]; the tree head names its epoch, size, root and the hash of the previous head; W witnesses cosign it. A member proves a credit, or anyone proves a penalty, with an inclusion proof of $\lceil \log_2 N \rceil$ hashes. Balances are a deterministic replay of the log, so the authority's balance can be checked by anyone holding it.

> PROPOSITION 4 (NO QUIET REWRITE)
> A witness cosigns a head only if its epoch is the next one the witness expects and its `prev` field equals the hash of the last head that witness cosigned. A head is accepted with more than $W/2$ cosignatures. Then two different heads for the same epoch cannot both be accepted unless some witness cosigned both, which an honest witness never does. *Proof.* Two majorities of W witnesses intersect; the common witness signed at most one head for that epoch. The log operator can stop publishing; it cannot show two members two histories. The proposition assumes no witness is faulty: with one faulty witness among three, two majorities can intersect only in that witness, which may sign two heads for one epoch. The rule above also checks head chaining, not that the new log's contents contain the old log's. A fixed witness roster with retained-prefix validation before signing is the correction path (corrected statement added in v1.1.1).

We chose a witnessed transparency log over a consensus chain on purpose. A public chain or a per-hop BFT round would put seconds or a consensus round between tokens. A permissioned BFT committee that orders only epoch heads is a compatible later step if THOX should not be the sole log operator; it changes who signs heads, not the request path.

4.6 The cloud plane

Subscribers also want models no pool of small devices can hold. The cloud plane reaches a hosted model through the same relay with the same tokens, bound to the `cloud` audience so a community token cannot be replayed there. The gateway

can read the prompt; that is what hosted inference is. What the plane adds is that the provider receives a request it can authorise and bill against THOX without learning which subscriber sent it, in the manner of [21, 23]. The response carries the label `cloud`, and the coordinator refuses any advertisement that lists a cloud-tagged model as contributed compute, which is ADR-025 enforced at the pool boundary [34].

4.7 Availability

A route needs one online, owner-idle contributor per remote stage. If each of n_s enrolled devices for a stage is independently available with probability a, a route with r-fold choice per stage across S remote stages exists with probability

$$P_{\text{route}} = \left[\sum_{k=r}^{n_s} \binom{n_s}{k} a^k (1-a)^{n_s-k} \right]^S$$

Device availability a	Devices per stage for one candidate (r = 1)	Devices per stage for a spare (r = 2)
0.50	8	12
0.67	5	8
0.80	4	6
0.90	3	4

Table 3. Calculated. Smallest enrolment per stage for a route to exist with probability at least 0.99 across 2 remote stages. Independence is an assumption; household devices sleep at the same hours.

5 Implementation

The virtual lab is 15 operating-system processes on one host that speak HTTP over loopback: authority, coordinator, relay, four contributor stage nodes, one auditor, a cloud gateway, the settlement ledger, three witnesses, and the requester's device. The model is SmolLM2-135M-Instruct [36] (135M parameters, 30 layers) at a pinned revision, split into three ten-layer stages. Each stage node loads the checkpoint and keeps only its own layer range (35.4M of 134.5M parameters). Cryptography is real: RSA blind signatures are implemented from RFC 9474 in Python integers and checked against the RFC's Appendix A.3 vector and against an independent library verifier; HPKE [22], Ed25519 [37] and RSASSA-PSS verification come from the `cryptography` library. Hops are stateless in the lab (each step recomputes the sequence), which makes reroute trivial and costs throughput we do not report.

The lab is virtual in one sense only: every party is a process, not a machine. It establishes that the protocol composes, that the refusals refuse, and that the

information each party receives is what the design says. It establishes nothing about THOX devices, wide-area links or throughput.

Req.	Exercised by	Result recorded
R1	Lab D (owner returns mid-request); simulation owner-delay bound	request moved off the owner's device at the next hop, output token-exact: True; simulated worst owner wait 32.0 s against a one-chunk bound of 32.0 s
R2	Lab B (tokens), Lab I (wire scan)	authority saw the account and no token value; coordinator, relay, stage nodes and gateway received no account on the wire (213 tokens issued)
R3	Lab B, Lab H	`local_only` at the coordinator: local_only_never_leaves_device; `mesh_only`: mesh_only_never_leaves_owner_devices; `confidential`: confidential_unavailable_no_attested_boundary; `community` class at the cloud gateway: privacy_class_forbids_cloud_plane
R4	Lab A, Lab B, Lab G	non-subscriber: not_a_subscriber; second redemption: double_spend; forged signature: invalid_token; over budget: budget_exhausted
R5	Experiment E1; Lab C, Lab F	digest audit exposed 102 of 102 cheated requests; lab fraud proof accepted, node suspended
R4, R6	Lab L	past the budget with the pool idle: issued as surplus (200); with every route leased: budget_exhausted; contributor tier as surplus: refused (429)
R6	Lab G	contributor's contributor-tier budget 13, consumer's 0; grant order b c c c b c b b b b b
R7	Lab A, Lab H	cloud-tagged advertisement: cloud_model_is_not_mesh_compute; cloud response label: cloud
R8	By construction	stateless hops; no KV state leaves a process in the lab. Not a test of the production KV graph.
R9	Experiment E4; Lab J	123 records, 3 of 3 cosignatures valid, replayed balances equal the authority's: True, forged head cosigned by 0 witnesses
R10	Lab D, Lab E, Lab F	killed contributor named in the error (C1), rerouted, output token-exact: True

Table 4. Requirement to test. Lab letters refer to Section 6.6. Every cell in the right column is read from the evidence file.

6 Evaluation

6.1 Setup and questions

All measurements ran on one Linux container (x86_64, 2 vCPU, 7 GB RAM, Python 3.13.18, PyTorch 2.15.0+cpu on CPU). The model is SmolLM2-135M-Instruct at revision `12fd25f7` [36], fp32, greedy decoding. Timings are medians after warm-up with the repetition count stated; nothing else ran on the host during a timed run.

Results are labelled *measured*, *virtual lab*, *simulated* or *calculated*, and the labels are not interchangeable. Five questions:

1. Which audit predicate exposes a cheating contributor, and how fast under sampling? (measured)
2. What does requester unlinkability cost? (measured)
3. What happens to fair admission when the pool cannot identify members? (simulated)
4. What does a blockchain-like settlement ledger cost, and where? (measured)
5. Does the whole protocol compose, and do its refusals refuse? (virtual lab)

6.2 Audit predicates on a real model

We split the model into three ten-layer stages inside one process, serialising the residual stream to fp32 bytes at each boundary and hashing those bytes into a receipt chain. A contributor runs the middle stage. For each of 17 prompts we generate 32 tokens honestly and under six cheats, and ask two auditors whether they would notice: one re-executes the request and compares output tokens, the other compares the boundary digest.

Baseline sanity. The staged pipeline reproduced the whole model's tokens on 17 of 17 prompts. Honest boundary digests were identical across two repeats (17 of 17) and between two threads and one thread (17 of 17), so on this host an honest contributor is never flagged. Staging cost nothing measurable: 1.53 s per request staged against 1.53 s whole (medians of 5 alternating passes; the difference, 0.0% in magnitude, is inside run-to-run spread). With key-value caching, a request moved 320 KiB across the two boundaries.

Cheat by the middle stage	Boundary-digest audit exposes	Token-exact audit exposes	Median first diverging token
skip 1 of 10 layers	17 / 17	17 / 17	token 0
skip 3 of 10 layers	17 / 17	17 / 17	token 0
skip the whole stage	17 / 17	17 / 17	token 0
stage weights rounded to fp16	17 / 17	0 / 17	never diverged
stage weights quantised to int8	17 / 17	16 / 17	token 6
0.1% Gaussian noise on the stage output	17 / 17	2 / 17	token 0

Table 5. Measured. Per-request detectability d for two audit predicates, 17 prompts, 32 greedy tokens each. The digest audit compares against an honest replay on the same host.

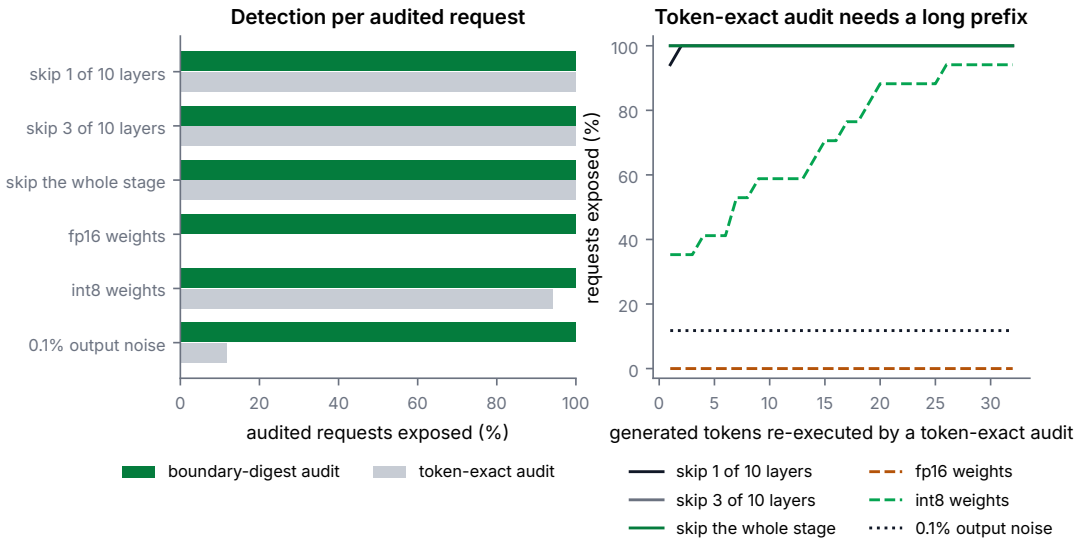


Figure 2. Left: share of audited requests that expose each cheat, by predicate. Right: share exposed by a token-exact audit as a function of how many generated tokens the auditor re-executes. Measured, 17 prompts.

The negative result. Comparing tokens exposes gross cheats immediately: any skipped layer changed the first or second token on every prompt. It does not expose economical ones. Rounding the stage's weights to fp16 changed no output token on any of the 17 prompts within 32 tokens, and a 0.1% output perturbation changed tokens on 2. Int8 weights were exposed on 16 of 17, but the median first divergence was token 6, so a short audit prefix misses most of them (Figure 2, right). These are exactly the substitutions a contributor has a reason to make, because they save memory or compute. MeshStack's certification gate for stage decomposition is token-exactness against the whole model (decision D25 [2]); it is the right gate for certifying an engine and the wrong predicate for auditing a stranger. The boundary digest exposed every cheat on every prompt.

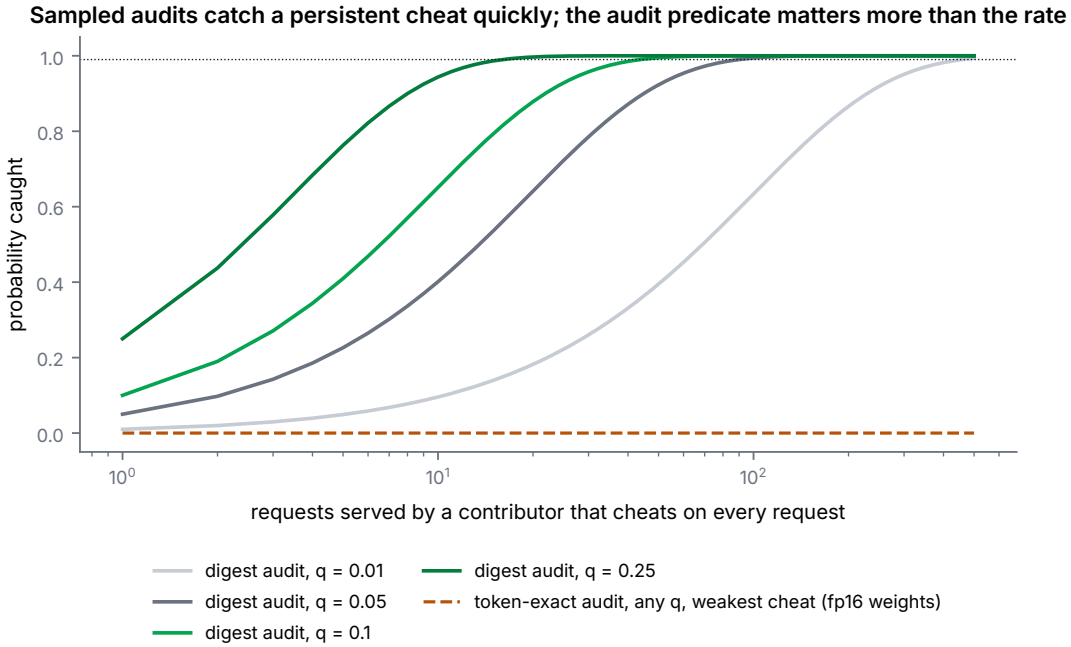


Figure 3. Calculated from Proposition 2 with the measured detectabilities: probability that a contributor cheating on every request has been caught, against requests served. Digest audit has d = 1 for all six cheats; the dashed line is the token-exact audit against its weakest cheat.

With the digest predicate, sampling is cheap. At an audit rate of 5% a persistent cheat is caught with 99% probability within 90 requests, at 10% within 44, at 25% within 17 (Figure 3). By Proposition 3, if a detected cheat forfeits 100 times what one cheated request earns, any audit rate above about 1% makes cheating unprofitable; at a ratio of 10 the rate must exceed about 9%. With the token predicate against an fp16 substitution, no audit rate suffices, because d was zero in our sample.

> RECOMMENDATION
>
> Receipts carry SHA-256 digests of each stage's input and output. An audit is a replay of the boundary by a node in the same determinism class. Token comparison is kept as a certification gate only. The stake a contributor forfeits should be at least 100 requests' worth of credit so that a 5% audit rate has a wide margin.

6.3 The cost of not being known

The blind-signature implementation reproduced RFC 9474 Appendix A.3 exactly: encoded message, blinded message, blind signature and final signature all matched (5 of 5 checks including verification by an independent library). At RSA-2048, over 200 runs, blinding took 0.64 ms on the client, the issuer's blind signature 7.25 ms, unblinding 0.02 ms and verification 0.12 ms; all 200 tokens verified. At RSA-4096 the issuer's cost rises to 47.7 ms (60 runs). The signing arithmetic here is plain Python integers, so these are upper bounds on a native

implementation. A token is one signature per request, obtained in advance, so none of this is on the token path.

Sealing a message to a recipient with HPKE adds 48 bytes and took 89 to 148 microseconds across payloads from 256 bytes to 64 KiB (200 runs each); opening took 55 to 91 microseconds. The cost is dominated by the key encapsulation, not the payload, and the medians are not monotonic in size at this scale. The envelope is paid once per hop. Against a hop that carries hundreds of kilobytes of activations and runs ten transformer layers, it is not the term that matters; the extra network hop through the relay is, and this paper does not measure it.

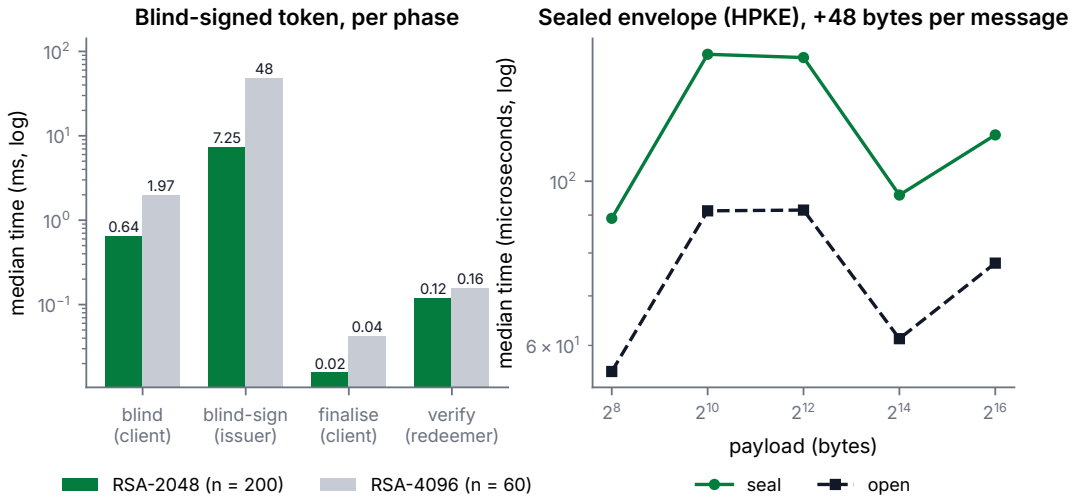


Figure 4. Measured. Left: per-phase cost of one blind-signed token (log scale). Right: HPKE seal and open time by payload size. Pure-Python big-integer arithmetic for the RSA phases; library HPKE.

6.4 Admission without identity (simulation)

The simulation has 12 contributor devices with abstract service rates of 8, 15 and 30 tokens per second, each unavailable while its owner is active (mean 20 minutes active, 40 idle, so availability 0.67), and 40 members of whom 12 own the devices and two submit 10 times the demand of the rest (34% of all demand). Five policies see identical arrival and availability traces: FIFO, least-served-first with counter lift (VTC-style, equal weights), the same with weights that grow with a member's contribution, the unlinkable form in which per-member token buckets gate entry to a FIFO pool, and that form with issuance indexed to pool load. 30 seeds per point, eight simulated hours each. The rates are scenario parameters, not device measurements.

Load	Policy	Contributors p95 wait	Other members p95 wait	Heavy members' share of service	Tokens served
0.6	FIFO	47 s	49 s	34.2%	1.91 M
0.6	least-served-first	19 s	18 s	34.2%	1.91 M
0.6	contribution-weighted	8 s	22 s	34.2%	1.91 M

Load	Policy	Contributors p95 wait	Other members p95 wait	Heavy members' share of service	Tokens served
0.6	fixed issuance budget (unlinkable)	3 s	464 s	9.3%	1.38 M
0.6	load-indexed budget (unlinkable)	13 s	17 s	34.2%	1.91 M
1.0	FIFO	1,498 s	1,473 s	34.8%	3.12 M
1.0	least-served-first	221 s	232 s	32.7%	3.12 M
1.0	contribution-weighted	15 s	300 s	32.7%	3.12 M
1.0	fixed issuance budget (unlinkable)	144 s	2,373 s	5.9%	2.19 M
1.0	load-indexed budget (unlinkable)	62 s	1,745 s	33.4%	3.12 M
1.2	FIFO	4,763 s	4,737 s	34.6%	3.19 M
1.2	least-served-first	635 s	644 s	21.1%	3.19 M
1.2	contribution-weighted	18 s	799 s	21.2%	3.19 M
1.2	fixed issuance budget (unlinkable)	272 s	4,639 s	5.2%	2.48 M
1.2	load-indexed budget (unlinkable)	194 s	4,502 s	26.4%	3.19 M

Table 6. Simulated. Medians over 30 seeds. Load is offered demand over expected pool capacity. Waits are to first service.

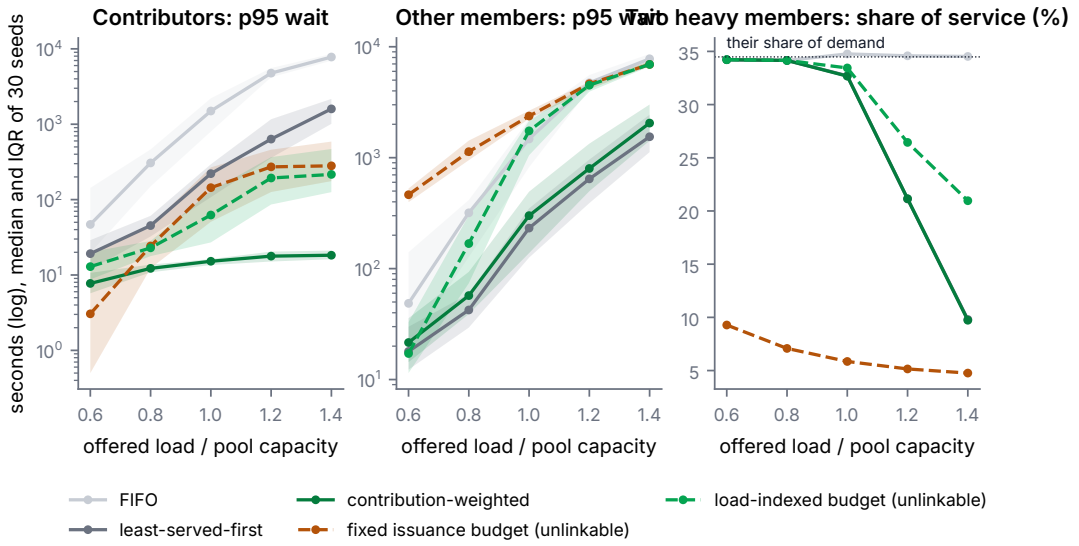


Figure 5. Simulated. p95 wait for contributors and for other members, and the heavy members' share of service, against load; median and interquartile range of 30 seeds. The dotted line is the heavy members' share of demand.

Three findings. First, with identity, contribution weighting does what it should: at load 1.2 contributors' p95 wait is 18 s against 635 s under equal weights and 4,763 s under FIFO, at a cost to other members (799 s against 644 s). Second, FIFO lets two heavy members take 35% of service at every load; counter-based policies cut that only once the pool saturates (21% at load 1.2).

The negative result. The unlinkable budget policy isolates heavy members at every load (9% of service at load 0.6), but as parameterised it is a poor scheduler. It is not work-conserving: at load 0.6 it served 1.38 M tokens where the others served 1.91 M, and ordinary members' p95 wait was 464 s against 18 s, because a member whose bucket is empty waits while devices idle. A fixed issuance rate is the wrong control.

Load-indexed issuance. Letting the authority issue past the budget while the pool reports an idle device and an empty queue removes the waste: at load 0.6 the policy served 1.91 M tokens (0.87 M of them as surplus) and ordinary members' p95 wait fell to 17 s. It is not a complete answer. Surplus issuance is first come, first served, so below saturation heavy members again take their share of demand (34%), which is harmless while capacity is spare; at load 1.2 they still take 26% against 21% under least-served-first, and ordinary members wait 4,502 s at p95 against 644 s. Unlinkable admission that matches counter-based fairness under overload remains open. Lab scenario L shows the mechanism working end to end: a member past their budget received a surplus token while the pool was idle and was refused while every route was leased.

Censoring (added in v1.1.1). The waiting times above cover requests that started. A never-started accounting over the same runs (arrival traces regenerated, started counts taken from the results above, no consistency failures across 750 runs) shows what the waits omit: at load 0.6 the fixed-budget policy leaves a median 27.7% of all requests never started, 80.3% for heavy members and 0.2% for ordinary members, while least-served-first leaves a median of 0% (59 of 139,921 requests pooled). The budget policy's isolation of heavy members is therefore stronger, and its service denial larger, than the waiting times alone suggest.

Owner protection. Across all 75,032 simulated owner returns, the longest an owner waited for their own device was 32.0 s, under the one-chunk bound of 32 s for a 256-token chunk on the slowest device (R1). The bound is a property of the chunk size and the device, not of the policy.

6.5 The settlement ledger

Signing a 239-byte receipt took 45 microseconds and verifying it 142 (200 runs); these are the only ledger-related costs paid per hop, and audits need them regardless. Building the Merkle tree for an epoch of 16,384 receipts took 15.0 ms (0.92 microseconds per receipt, 30 runs). 5 witness cosignatures on the head took

208 microseconds to produce and 561 to verify. An inclusion proof at that size is 14 hashes, 448 bytes, verified in 8.4 microseconds.

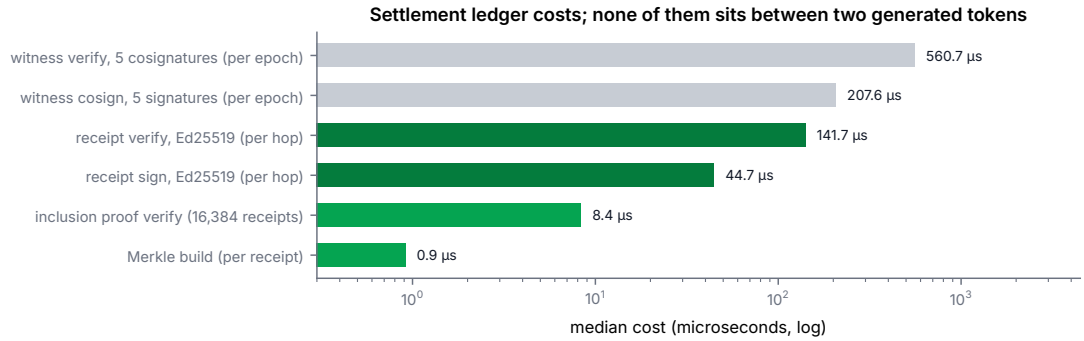


Figure 6. Measured. Ledger cost components on a log scale. Green bars are per hop or per receipt; grey bars are once per epoch.

The structure gives what a chain would be used for here: an ordered, append-only, publicly verifiable record that the operator cannot fork unnoticed (Proposition 4) and from which every balance can be recomputed. It does so with no work between two generated tokens. What it does not give is operator-independent liveness: the log operator can stop. If that matters, a small committee can order epoch heads; the request path does not change.

6.6 Virtual lab: does it compose?

The lab ran the full protocol across 15 processes (2 vCPU host) and recorded one evidence file. Table 7 lists the 12 scenarios. Community-plane requests were token-identical to the whole model on 8 of 8 prompts with 512 signed receipts verified by the device, and all 32 hops of one honest request were replayed by the auditor in a separate process with every digest matching, so cross-process replay raised no false alarm.

	Scenario	What the evidence file records	
A	Enrolment	5 devices enrolled with authority certificates and signed advertisements; self-signed certificate refused (no_authority_certificate); non-subscriber refused (not_a_subscriber); advertisement listing a cloud-tagged model refused (cloud_model_is_not_mesh_compute)	pass
B	Tokens	valid token redeemed for a signed lease; second redemption double_spend; forged signature invalid_token; cloud-audience token at the pool invalid_token; over-budget issuance budget_exhausted; `local_only` request local_only_never_leaves_device	pass
C	Inference	8 / 8 requests token-exact; 512 receipts verified; 32 honest hops audited, all match	pass
D	Owner returns	owner of B1 became active at step 8; next hop refused (owner_active); request continued on B2; output token-exact; a new lease while the owner was active avoided the device; it rejoined when the owner left	pass

	Scenario	What the evidence file records	
E	Contributor dies	process C1 killed at step 12; upstream stage reported downstream_failed naming C1; rerouted to C2; output token-exact	pass
F	Cheating contributor	digest audit exposed noise, fp16 and skipped-layer cheats at the first audited hop, while the noise and fp16 cheats left the unaudited output token-exact; with a sampled audit at q = 0.25, the cheat was caught at step 1, the fraud proof accepted, the node suspended, its owner's credit set to zero, and the request restarted on another node with token-exact output	pass
G	Admission	contributor's contributor-tier budget 13 tokens, consumer's 0; 8 base-tier and 4 contributor-tier requests granted in the order b c c c b c b b b b b b; heavy member's extra issuance budget_exhausted	pass
H	Cloud plane	request served through the relay with label `cloud` and the notice that the provider can read the prompt; token replay double_spend; community-class request privacy_class_forbids_cloud_plane	pass
I	Information flow	across 205 MB received by the relay, 90 MB by the busiest stage node and everything received by the coordinator, auditor and gateway, no consumer account, token value or prompt text appeared on the wire; the authority received accounts and no token value	pass
K	Replica route	a contributor joined mid-run holding both remote layer ranges (70.8M parameters) and was leased as a whole replica; 62 of 62 requests completed across both topologies, all token-exact; replica hops audited, all match (timing in Section 6.7)	pass
L	Load-indexed issuance	member past budget, pool idle: token issued and marked surplus; every route leased: budget_exhausted; routes released: issued again; contributor-tier surplus refused	pass
J	Ledger	123 coordinator-signed records sealed in one epoch; 3 of 3 witness cosignatures valid; balances replayed from the log equal the authority's; inclusion proof for the penalty record valid (7 hashes) and invalid after a one-character change; a forged head for the same epoch cosigned by 0 witnesses; next epoch chains to the first	pass

Table 7. Virtual lab. Every cell is generated from lab/evidence/evidence.json. A scenario passes only if every listed condition held.

Two observations from the lab belong in the design. First, scenario F reproduced the audit finding end to end: a contributor that perturbed or down-cast its output produced a response the requester could not distinguish from an honest one, and was still caught at the first audited hop. Second, scenario I is a statement about the wire, not about knowledge. Stage nodes decrypt activations and the gateway decrypts the prompt, by design. The scan shows that the relay, which holds no key, and the coordinator, which holds only its own, receive nothing readable that identifies a consumer.

Lab timings are not performance results. A hop took a median of 145 ms and a 32-token request 7.5 s, on stateless hops that recompute the whole sequence at every step, over loopback, with 15 processes sharing two cores.

6.7 Topology: whole replica against pipeline

Measured in the lab. In scenario K a third contributor joined mid-run holding both remote layer ranges, and the requester alternated between the two-stage pipeline route and the whole-replica route in ABBA order: 30 trials each of 8 generated tokens, one labelled warm-up per topology kept in the record, nothing discarded. The mean remote round trip per token was 134.5 ms for the pipeline (95% bootstrap interval 132.5 to 136.5; trial-mean CV 4.3%) and 124.4 ms for the replica (122.7 to 126.2; CV 4.0%), a reduction of 7.5%. The intervals do not overlap and both CVs are under the 10% dispersion condition of decision D37, which is necessary and not sufficient for a claim-grade number: this is one host, loopback, a small model and stateless hops. The saving here is one sealed hop and one receipt; the layers computed are the same.

Correction (v1.1.1). The comparison above gave each route different prompts, so the 7.5% figure is superseded. A matched-prompt rerun inside this pipeline (30 pairs, the same prompt on both routes, order alternating, auditing kept out of the timed requests, every attempt token-exact, recorded in exp/results/erratum_rerun.json) measured 126.4 ms per remote call for two contributors in sequence against 118.0 ms for one contributor holding both remote ranges on a quiet host: a paired difference of 8.35 ms (95% interval 6.27 to 10.27), 6.6% lower, with the single contributor faster in 27 of 30 pairs. An earlier run on a shared host agreed in direction (131.5 against 124.6 ms, 5.2% lower). The conclusion stands at the corrected size; the v1.1.0 figure does not.

Calculated for real links. The companion paper models one serial decode stream as compute plus per-link delay plus bytes over goodput [3]. We re-derived its four-stage example and obtained the same cycle times (44.39, 83.93, 167.86, 319.32 ms). Applying the same form to this design, with 40 ms of assumed total compute per token, a sealed fp32 position of 2,352 bytes for the test model, 303 bytes per signed receipt, six link traversals per token for the pipeline through the relay and four for a replica through the relay, gives Table 8. On a wide-area link the relay and the extra stage dominate: delay, not payload, is the term that matters at this hidden size.

Link assumption	One-way delay, goodput	Pipeline via relay	Whole replica via relay	Replica advantage
Nearby LAN	1 ms, 1,000 Mbps	46.1 ms (21.68 tok/s)	44.1 ms (22.69 tok/s)	5%
Metro	10 ms, 100 Mbps	101.3 ms (9.87 tok/s)	80.8 ms (12.38 tok/s)	25%
WAN	30 ms, 50 Mbps	222.5 ms (4.49 tok/s)	161.6 ms (6.19 tok/s)	38%
Constrained WAN	60 ms, 10 Mbps	412.7 ms (2.42 tok/s)	288.0 ms (3.47 tok/s)	43%

Table 8. Calculated, not measured. One sequence, no queueing, jitter, batching or speculation. The compute figure is an assumption held constant; it is not a THOX device rate. A hosted whole model that streams tokens pays the link once per request, not once per token.

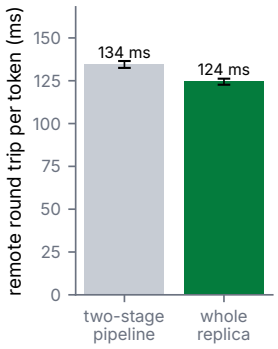


Figure 7. Left: measured remote round trip per token in this lab, pipeline against whole replica, mean and 95% bootstrap interval of 30 trials each. Centre: calculated single-stream rate by link class (Table 8). Right: the companion lab's inter-token interval by configuration, as retained by its authors and as reproduced on this host.

> RECOMMENDATION
> Route to a whole replica when one eligible device can hold the remote layers. Admit a cross-device partition only for a model that no single eligible device can hold, within a nearby group first, and only after measuring it. This matches the companion paper's default and the earlier household report's sequencing [3, 4].

6.8 The companion paper and lab: reproduction and reconciliation

The companion work [3] approaches the same problem from the governance side: who may process what, under which consent, with which evidence. Its lab runs six worker processes with mutual TLS and pinned certificates, per-call HMAC grants bound to the request body, tenant namespaces, transactional settlement and a signed, hash-linked receipt log, on an untrained 241,280-parameter float64 model under one trusted operator. Ours assumes hostile contributors and an anonymous requester on a trained model. The two labs test different things and neither subsumes the other.

Reproduction. We ran the companion lab unmodified on this host with its pinned dependencies under Python 3.12.3: 62 of 62 tests passed (43 unit, 19 integration). Its full campaign completed 150 of 150 trials plus 10 warm-ups, every trial numerically equal to its reference, 5 of 5 policy and fault scenarios passed, and the signed log verified with 164 receipts, the same count its authors retained. The

experiment plan hash and the model fingerprint are identical to the retained run, so the same model bits and the same plan ran on a different machine.

Configuration	Authors' host: mean inter-token	CV	This host: mean inter-token	95% interval	CV	Numerically equal
replica 0ms	16.62 ms	64.3%	12.80 ms	12.44 to 13.19	8.3%	30 / 30
pipeline 0ms	79.52 ms	16.5%	62.15 ms	60.93 to 63.40	5.5%	30 / 30
pipeline 5ms	100.77 ms	10.3%	85.86 ms	84.47 to 87.30	4.7%	30 / 30
pipeline 15ms	148.98 ms	23.7%	128.12 ms	126.81 to 129.53	3.1%	30 / 30
pipeline audit paused	81.83 ms	20.3%	62.59 ms	60.97 to 64.28	7.4%	30 / 30

Table 9. Companion lab timing, 30 trials per configuration, as retained by its authors (Python 3.12.14) and as reproduced here (Python 3.12.3, 2 vCPU). Observed on one host each; not a performance claim for either.

What the reproduction adds. The ordering and the effect of injected delay reproduce: the whole replica is several times faster per token than the four-stage pipeline, and each added wait per call raises the interval. The dispersion does not reproduce: the authors reported that every configuration exceeded the 10% CV condition (up to 64%), while on this host every configuration is under it (at most 8.3%). The noise was a property of their host, not of the lab. Neither run qualifies a performance statement. The comparison of an active and a paused audit writer also reproduces as no detectable difference (62.15 ms against 62.59 ms, overlapping intervals), which supports both papers' position that settlement belongs off the token path.

A portability defect. Under Python 3.13 the companion lab's 43 unit tests pass and its integration tests cannot start: the default TLS context in 3.13 verifies strictly and rejects the lab's generated certificates for a missing Authority Key Identifier. The lab documents Python 3.12 and works there. The certificates should carry subject and authority key identifiers before the lab is used on a current interpreter.

Topic	Companion paper	This paper	Status
Trust model	one trusted operator; contributors approved	hostile contributors; anonymous requester	complementary; both labs kept, never merged into one result
Model in the lab	untrained 241,280-parameter float64 fixture	trained 135M-parameter SmolLM2-135M-Instruct, fp32	complementary

Topic	Companion paper	This paper	Status
Requester identity	tenant-scoped pseudonym known to the coordinator	blind token; coordinator never learns the account	**settled 2026-11-10, composed.** The tenant pseudonym is the accounting and quota identity at the coordinator; the blind token is the requester-facing execution credential, so fairness is enforced at issuance against the pseudonym and execution stays unlinkable (Section 6.4).
Default topology	whole replicas first; partitions only where measured	pipeline assumed in v1.0.0	adopted from the companion in this edition; measured in Section 6.7
Admission	plan-wide eligibility rule and plan digest	per-hop lease checks	adopted; the lease now carries a plan digest
Confidential context	refused without an attested boundary	not modelled in v1.0.0	adopted; refused in lab scenario B
Transport between hosts	mutual TLS with pinned certificates, HMAC grant per call	HPKE sealing per message, signed lease	both valid; production should use the tunnel and identities MeshStack already has, plus per-message sealing through the relay
Integrity of a host's work	receipt records the action; not proof of honest computation	boundary-digest replay audit and fraud proofs	this paper adds the audit; the companion states the gap
Settlement	transactional outbox, signed hash-linked batches, retained checkpoint	Merkle epochs, witness cosignatures, replay	compatible. The companion's outbox and idempotency should feed this paper's witnessed log; the companion notes a signed prefix cannot detect a dropped suffix without an outside checkpoint, which witnesses provide
Contribution accounting	credits and settlement are a new product decision	non-transferable credits raise token budget and tier	agree that it needs a decision; D20 stands until then
Byline	Craig Ross and Tommy Xaypanya	Craig Ross and Tommy Xaypanya	**settled 2026-11-10.** One roster for both papers: Craig Ross first author, Tommy Xaypanya corresponding author; this edition carries it

Table 10. Reconciliation of the two works. Nothing was merged silently; where they differ, the difference and its owner are named.

7 Discussion and limitations

- **Content is not private from the executing contributor.** A stage node receives activations, and published attacks invert activations to text [9]. The lab's wire

scan shows what crosses the wire unsealed, not what a recipient can decrypt and compute on. The community plane is for requests whose owner accepts that exposure; the default class for anything personal should remain `local_only`, as the security model's operator checklist already says [10].

- **The digest audit is bound to a determinism class.** Digests matched across repeats, across one and two threads, and across separate processes on this host. They were not tested across CPU models, accelerators, operating systems or engine builds, where bit-identical floating point is not expected. Decision D31 already limits bit-exact promises to a pinned engine, hardware class and batch shape [2]. Until a tolerant commitment such as TOPLOC [17] is evaluated, an auditor must be in the contributor's class, which constrains who can audit whom.
- **Host, model and scale.** One 2-vCPU x86_64 container; SmolLM2-135M-Instruct at 135M parameters; 17 prompts of 32 greedy tokens; 5 timing passes for the staging overhead; 200 repetitions for cryptographic timings (60 at RSA-4096, 30 for the largest Merkle batch). Detection rates over 17 prompts are coarse: zero of 17 does not mean zero.
- **The lab is one host.** No THOX device, no second machine, no wide-area link, no WireGuard tunnel, no real relay operator. Hops are stateless and recompute the sequence at every step, so lab latency says nothing about a KV-cached engine. No throughput or latency figure in this paper is a product claim; decision D37 forbids them until the benchmark gate passes [2].
- **Admission results are a simulation** with abstract service rates, exponential owner activity and independent members. Real households are correlated: devices sleep at night together. The budget policy's parameters were chosen once and not tuned.
- **The lab control services are not a security boundary.** Independent probes of the lab services found defects the paper did not list: blind tokens replayable after a coordinator restart; diagnostics readable without authentication; enrollment able to claim another owner's identity; receipts creditable twice or against the wrong lease; a member able to close or reroute another member's lease; replayed owner-state and stale capacity advertisements accepted; unknown privacy labels admitted; audit findings accepted without binding to the job they describe; and settlement not checking the plan digest. These are limits of the lab evidence; the fixture is not a hardened service, and the fixes belong to the engineering roadmap, not to this paper's claims.
- **Collusion and traffic analysis are out of scope.** A relay that colludes with the coordinator links address to request; this is the standard OHTTP assumption [21] and requires the two to be operated by different parties. Request timing and size are visible to the relay. With few members online, the anonymity set is small regardless of cryptography.

- **Sybil resistance is economic.** A subscription is the cost of an identity; per-account budgets and device limits bound what one identity can draw. A wealthy adversary can buy many subscriptions. Auditors are assumed honest; a dishonest auditor could frame a contributor, which is why a fraud proof should be re-checkable by a second auditor before a permanent penalty. The lab accepts a single finding.
- **Advertised capacity is self-reported.** A device can overstate what it holds. The lab certifies only the model fingerprint string; binding a layer package to a measured hash at enrolment is the production requirement (decisions D16 and D17).
- **Topology timing.** The replica-against-pipeline comparison is 30 trials per arm on one host over loopback. Its low dispersion is a property of this host on this day; the companion lab showed the opposite on its authors' host. The network figures in Table 8 are calculated from assumptions.
- **Two labs, not one.** The companion lab and this lab share no code and answer different questions. A pass in one says nothing about the other. Their consolidation into the canonical MeshStack lines is future work.
- **Not built.** Device attestation, key rotation and token epochs, the production coordinator and Supabase store, KV-cached stages with migration, the own-mesh plane inside this lab, a second auditor, a BFT committee for heads, billing for the cloud plane, and every client surface.
- **Three prior decisions bear on this design and are the founders' to keep or change:** D20 (no payments market: we use non-transferable credits), D43 (no cross-principal cache until a security review: we share none), and ADR-025 (cloud tags are never mesh compute: we enforce it and add a separate, labelled cloud plane).
- **Not peer reviewed.** This is a preprint prepared by THOX.ai Research with AI assistance (Appendix C).

8 Conclusion

A community pool for inference needs five things at once: owners who never lose their own device, requesters the pool cannot identify, cheats that are caught and cost more than they earn, access that follows contribution without a currency, and bookkeeping that does not sit between tokens. Each has a known building block. This paper assembles them on MeshStack, runs the assembly end to end in a virtual lab, and reports the one result that changed the design: the natural audit, comparing output tokens, let an fp16 substitution through on every one of 17 prompts, while a digest of the stage boundary caught everything. It also states plainly what the pool does not give: content privacy from the device that does the work. With that limit on the label, pooled hardware plus an anonymous path to

hosted models is a credible way to give subscribers inference they could not otherwise afford.

9 Roadmap

Ordered by what unblocks what.

1. **Decide D20, D43 and ADR-025 for the community scope, the requester-identity default (pseudonym or blind token) and one author roster.** The design assumes the three decisions stand.
2. **One lab.** Bring the companion lab's mutual TLS, per-call grants, tenant namespaces and transactional outbox together with this lab's tokens, audits and witnessed ledger on the canonical lines; fix the certificate identifiers.
3. **Two-machine lab.** Run the same roles on two fleet hosts over a WireGuard tunnel, with the relay on a third party's host. Exit evidence: the same evidence file with real addresses in the wire scan.
4. **Determinism classes.** Measure boundary-digest equality across the device classes that will contribute; where it fails, evaluate a tolerant commitment and report its false-accept rate against the six cheat classes.
5. **KV-cached stages.** Replace stateless hops with the stage-server engine and its token-log replay; re-run scenarios D, E and F.
6. **Carry the pool into the canonical lines.** Extend the Auto Device Role Mode tables with a community scope, add tier keys to the authority, and land the ledger beside the coordinator in ttracx/sadie-mesh.
7. **Second auditor and appeal.** Require two independent findings before a permanent penalty.
8. **Device benchmark under the claims register.** Thirty runs per configuration on THOX devices before any performance sentence is written.
9. **External review** of the token, relay and ledger construction before any public release.

A Appendix A. Reproducibility

Environment: Linux container, x86_64, 2 vCPU, 7 GB RAM; Python 3.13.18; PyTorch 2.15.0+cpu (CPU wheel); transformers 5.22.2; cryptography 50.0.1; NumPy; Matplotlib. Node 22 with katex 0.16.11, @fontsource/inter 5.1.0, @fontsource/jetbrains-mono 5.1.1 and Playwright for rendering. Greedy decoding throughout; the model revision is pinned.

Commands (from the artifact root)

```
pip install torch --index-url https://download.pytorch.org/whl/cpu && pip install
transformers safetensors cryptography numpy matplotlib pypdf pillow
python3 exp/audit_experiments.py         # downloads the model; about 10 minutes on 2
vCPU
(cd exp && python3 audit_timing.py)      # five alternating timing passes
python3 exp/token_experiments.py         # RFC 9474 known-answer test, blind RSA and
HPKE timings
python3 exp/fairness_sim.py              # 30 seeds x 5 loads x 4 policies
python3 exp/ledger_experiments.py
python3 -m lab.run_lab                   # the virtual lab; writes
lab/evidence/evidence.json; exit code 0 only if every scenario passes
# companion lab (needs Python 3.12): unzip
companion/THOX_MeshStack_Virtual_Lab_v1.0.zip; pip install -r lab/requirements.txt
#   python -m unittest discover -s tests && python -m mesh_lab experiment --output
evidence/<name> --blocks 30
python3 exp/analyze.py                   # writes exp/results/summary.json
python3 wp/figures.py && python3 wp/build.py && node wp/katex_render.cjs
wp/paper.src.html wp/paper.html
node wp/pdf.cjs wp/paper.html wp/cover.pdf wp/body.pdf && python3 wp/merge_pdf.py &&
python3 wp/build_demo.py
```

Lab evidence root for this edition: Merkle root of the settlement ledger's first epoch `14ea8aed0f41456a688ab65449ad33be9a1cf117604ac3bac4f15b06e39aff13`. The artifact contains every script, the raw results, summary.json, the lab source, its evidence file, per-process event logs and transcript, the figure sources and the built outputs.

B Appendix B. Notation

Symbol	Meaning
n, e, d	RSA modulus, public and private exponent of a tier key
$m, r, z, \hat{s}, s$	token message, blinding factor, blinded message, blind signature, token signature
Q	service quantum: generated tokens one token authorises
σ_τ, w_τ	service granted to tier and tier weight
B_m, B_0, β, c_m	member budget, base budget, bonus rate, verified service contributed
q, d	audit rate; probability an audit exposes a given cheat
g, L	gain per cheated request; loss on detection
W, N	witnesses; ledger entries in an epoch
a, n_s, r, S	device availability, devices enrolled per stage, required candidates per stage, remote stages

C Appendix C. Authorship and disclosure

This preprint is authored by Craig Ross and Tommy Xaypanya, THOX.ai LLC, and issued by THOX.ai Research. The corresponding and accountable author is Tommy Xaypanya, Chief Technology Officer and Co-Founder (ORCID 0009-0000-4425-9565). The roster was settled by the founders on 2026-11-10 and is the roster for any external submission of either paper. The companion paper and lab were prepared with a different AI system (ChatGPT, OpenAI) and are cited, reproduced and reconciled here, not merged. Generative AI assistance: the repository review, literature retrieval, experiment and lab code, analysis, figures and draft text were produced by an AI system (Claude, Anthropic) operating as the THOX digital human Odette Lark under the direction of the corresponding author. Every arXiv entry was resolved against the arXiv API, every RFC against the RFC Editor's record and every venue page was opened on 2026-11-10; every number is generated from the experiment and lab outputs. No AI system is an author. Conflicts: the author is a THOX.ai principal and MeshStack is a THOX.ai product; this is an interested-party technical account. External submissions proceed under the founders' standing approval recorded on 2026-11-04.

References

[1] THOX.ai. *Auto Device Role Mode (THOX-MESH-ROLE-001, version 0.1.1)*. ttracx/meshstack-backend, docs/auto-device-role-mode.md, 2026-11-02 (internal engineering preview).

[2] THOX.ai. *Decisions of record D1 to D49*. ttracx/thox-meshstack-inference, DECISIONS.md, commit 7e86cec, read 2026-11-10 (internal).

[3] Craig Ross, Tommy Xaypanya. *MeshStack Community Compute: Privacy governed shared hardware and distributed AI*. THOX.ai LLC research paper THOX-RP-COMMUNITY-2026-001, version 1.0, 10 November 2026, with the THOX MeshStack virtual lab v1.0 (confidential, prepared for author review; PDF SHA-256 e202e7d0...14fe).

[4] Craig Ross, Tommy Xaypanya. *THOX MeshStack: Capacity-Priced Pipeline Sharding, Grant-Gated KV Transfer and Lease-Fenced Device Roles for a Private Household Mesh*. THOX.ai Research technical report v1.0.1, 2026-11-03, Zenodo, doi:10.5281/zenodo.23919580. https://doi.org/10.5281/zenodo.23919580

[5] David P. Anderson. *BOINC: A Platform for Volunteer Computing*. arXiv:1903.01699v1, first posted 2019-03-05, version inspected 2019-03-05. https://arxiv.org/abs/1903.01699

[6] Luis F. G. Sarmenta. *Sabotage-Tolerance Mechanisms for Volunteer Computing Systems*. ACM/IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01), Brisbane, May 2001. https://groups.csail.mit.edu/cag/bayanihan

[7] Alexander Borzunov, Dmitry Baranchuk, Tim Dettmers, Max Ryabinin, Younes Belkada, Artem Chumachenko, Pavel Samygin, Colin Raffel. *Petals: Collaborative Inference and Fine-tuning of Large Models*. arXiv:2209.01188v2, first posted 2022-09-02, version inspected 2023-03-02. https://arxiv.org/abs/2209.01188

[8] Alexander Borzunov, Max Ryabinin, Artem Chumachenko, Dmitry Baranchuk, Tim Dettmers, Younes Belkada, Pavel Samygin, Colin Raffel. *Distributed Inference and Fine-tuning of Large Language Models Over The Internet*. arXiv:2312.08361v1, first posted 2023-12-13, version inspected 2023-12-13. https://arxiv.org/abs/2312.08361

[9] Wenjie Qu, Yuguang Zhou, Yongji Wu, Tingsong Xiao, Binhang Yuan, Yiming Li, Jiaheng Zhang. *Prompt Inversion Attack against Collaborative Inference of Large Language Models*. arXiv:2503.09022v3, first posted 2025-03-12, version inspected 2025-05-02. https://arxiv.org/abs/2503.09022

[10] THOX.ai. *Security model: KV cache, grants and the mesh*. ttracx/thox-meshstack-inference, docs/SECURITY.md, commit 7e86cec, read 2026-11-10 (internal).

[11] Chris Tong, Youhe Jiang, Gufeng Chen, Tianyi Zhao, Sibian Lu, Wenjie Qu, Eric Yang, Lynn Ai, Binhang Yuan. *Parallax: Efficient LLM Inference Service over Decentralized Environment*. arXiv:2509.26182v1, first posted 2025-09-30, version inspected 2025-09-30. https://arxiv.org/abs/2509.26182

[12] Zonghang Li, Tao Li, Wenjiao Feng, Rongxing Xiao, Jianshu She, Hong Huang, Mohsen Guizani, Hongfang Yu, Qirong Ho, Wei Xiang, Xue Liu. *Prima.cpp: Fast 30-70B LLM Inference on Heterogeneous and Low-Resource Home Clusters*. arXiv:2504.08791v3, first posted 2025-04-07, version inspected 2026-07-04. https://arxiv.org/abs/2504.08791

[13] KD Conway, Cathie So, Xiaohang Yu, Kartin Wong. *opML: Optimistic Machine Learning on Blockchain*. arXiv:2401.17555v2, first posted 2024-01-31, version inspected 2024-02-05. https://arxiv.org/abs/2401.17555

[14] Yue Zhang, Shouqiao Wang, Sijun Tan, Xiaoyuan Liu, Ciamac C. Moallemi, Raluca Ada Popa. *Proof of Sampling: A Nash Equilibrium-Based Verification Protocol for Decentralized Systems*. arXiv:2405.00295v3, first posted 2024-05-01, version inspected 2025-05-31. https://arxiv.org/abs/2405.00295

[15] Arther Tian, Alex Ding, Frank Chen, Alan Wu, Aaron Chan, Bruce Zhang. *Design and Evaluation of Cost-Aware PoQ for Decentralized LLM Inference*. arXiv:2512.16317v1, first posted 2025-12-18, version inspected 2025-12-18. https://arxiv.org/abs/2512.16317

[16] Yifan Sun, Yuhang Li, Yue Zhang, Yuchen Jin, Huan Zhang. *SVIP: Towards Verifiable Inference of Open-source Large Language Models*. arXiv:2410.22307v3, first posted 2024-10-29, version inspected 2026-01-31. https://arxiv.org/abs/2410.22307

[17] Jack Min Ong, Matthew Di Ferrante, Aaron Pazdera, Ryan Garner, Sami Jaghouar, Manveer Basra, Max Ryabinin, Johannes Hagemann. *TOPLOC: A Locality Sensitive Hashing Scheme for Trustless Verifiable Inference*. arXiv:2501.16007v2, first posted 2025-01-27, version inspected 2025-05-30. https://arxiv.org/abs/2501.16007

[18] A. Davidson, J. Iyengar, C. A. Wood. *The Privacy Pass Architecture*. RFC 9576, IETF, June 2024. https://www.rfc-editor.org/rfc/rfc9576

[19] S. Celi, A. Davidson, S. Valdez, C. A. Wood. *Privacy Pass Issuance Protocols*. RFC 9578, IETF, June 2024. https://www.rfc-editor.org/rfc/rfc9578

[20] F. Denis, F. Jacobs, C. A. Wood. *RSA Blind Signatures*. RFC 9474, IRTF, October 2023. https://www.rfc-editor.org/rfc/rfc9474

[21] M. Thomson, C. A. Wood. *Oblivious HTTP*. RFC 9458, IETF, January 2024. https://www.rfc-editor.org/rfc/rfc9458

[22] R. Barnes, K. Bhargavan, B. Lipp, C. Wood. *Hybrid Public Key Encryption*. RFC 9180, IRTF, February 2022. https://www.rfc-editor.org/rfc/rfc9180

[23] Apple Security Engineering and Architecture, User Privacy, Core Operating Systems, Services Engineering, and Machine Learning and AI. *Private Cloud Compute: A new frontier for AI privacy in the cloud*. Apple Security Research blog, 10 June 2024, accessed 2026-11-10. https://security.apple.com/blog/private-cloud-compute/

[24] Roger Dingledine, Nick Mathewson, Paul Syverson. *Tor: The Second-Generation Onion Router*. 13th USENIX Security Symposium, 2004. https://www.usenix.org/conference/13th-usenix-security-symposium/tor-second-generation-onion-router

[25] John X. Morris, Volodymyr Kuleshov, Vitaly Shmatikov, Alexander M. Rush. *Text Embeddings Reveal (Almost) As Much As Text*. arXiv:2310.06816v1, first posted 2023-10-10, version inspected 2023-10-10. https://arxiv.org/abs/2310.06816

[26] John X. Morris, Wenting Zhao, Justin T. Chiu, Vitaly Shmatikov, Alexander M. Rush. *Language Model Inversion*. arXiv:2311.13647v1, first posted 2023-11-22, version inspected 2023-11-22. https://arxiv.org/abs/2311.13647

[27] Sayedeh Leila Noorbakhsh, Hossein Khalili, Nader Sehatbakhsh. *Defense Against Prompt Inversion Attacks: An Information-Theoretic Approach for LLM Collaborative Inference*. arXiv:2606.11592v1, first posted 2026-06-10, version inspected 2026-06-10. https://arxiv.org/abs/2606.11592

[28] Ye Dong, Wen-jie Lu, Yancheng Zheng, Haoqi Wu, Derun Zhao, Jin Tan, Zhicong Huang, Cheng Hong, Tao Wei, Wenguang Chen. *PUMA: Secure Inference of LLaMA-7B in Five Minutes*. arXiv:2307.12533v3, first posted 2023-07-24, version inspected 2023-09-26; 2025. https://arxiv.org/abs/2307.12533

[29] Marcin Chrapek, Marcin Copik, Etienne Mettaz, Torsten Hoefler. *Confidential LLM Inference: Performance and Cost Across CPU and GPU TEEs*. arXiv:2509.18886v1, first posted 2025-09-23, version inspected 2025-09-23. https://arxiv.org/abs/2509.18886

[30] Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E. Gonzalez, Ion Stoica. *Fairness in Serving Large Language Models*. arXiv:2401.00588v2, first posted 2023-12-31, version inspected 2024-06-05. https://arxiv.org/abs/2401.00588

[31] Shiyi Cao, Yichuan Wang, Ziming Mao, Pin-Lun Hsu, Liangsheng Yin, Tian Xia, Dacheng Li, Shu Liu, Yineng Zhang, Yang Zhou, Ying Sheng, Joseph Gonzalez, Ion Stoica. *Locality-aware Fair Scheduling in LLM Serving*. arXiv:2501.14312v1, first posted 2025-01-24, version inspected 2025-01-24. https://arxiv.org/abs/2501.14312

[32] Dev Bali, Soujanya Ponnapalli, Yichuan Wang, Natacha Crooks, Scott Shenker, Matei Zaharia. *Token Latency Fairness: Performance Isolation for Multi-Tenant LLM Serving*. arXiv:2609.18112v1, first posted 2026-09-16, version inspected 2026-09-16. https://arxiv.org/abs/2609.18112

[33] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, Ion Stoica. *Dominant Resource Fairness: Fair Allocation of Multiple Resource Types*. NSDI 2011. https://www.usenix.org/conference/nsdi11/dominant-resource-fairness-fair-allocation-multiple-resource-types

[34] THOX.ai. *ADR-025: Ollama Cloud tags are not local mesh compute*. ttracx/meshstack-desktop, docs/architecture_decisions.md, commit b2b6e3f, read 2026-11-10 (internal).

[35] B. Laurie, E. Messeri, R. Stradling. *Certificate Transparency Version 2.0*. RFC 9162, IETF, December 2021. https://www.rfc-editor.org/rfc/rfc9162

[36] Hugging Face TB Research. *HuggingFaceTB/SmolLM2-135M-Instruct model repository, revision 12fd25f7*. Hugging Face, accessed 2026-11-10. https://huggingface.co/HuggingFaceTB/SmolLM2-135M-Instruct

[37] S. Josefsson, I. Liusvaara. *Edwards-Curve Digital Signature Algorithm (EdDSA)*. RFC 8032, IRTF, January 2017. https://www.rfc-editor.org/rfc/rfc8032