cifra

One model. A thousand ways to encrypt it. Cifra pays for the fastest circuit that still computes it correctly — and cannot trade privacy for speed.

Cifra is a Bittensor subnet for private machine-learning inference. An owner publishes a plaintext reference model; miners compete to compile it into the fastest fully-homomorphic circuit that still reproduces it. Validators re-compile every submission themselves, enforce 128-bit security so no one can buy speed by weakening the cryptography, gate correctness against deterministic checks every validator computes to the same digit, and then rank the survivors by the one thing a buyer actually pays for — measured encrypted-inference latency — with rewards flowing geometrically down the leaderboard. The prize is not a promise about a running server — it is an artifact anyone can deploy.

PROTOCOL PAPER · V0.4 · 2026 · PRE-MAINNET DRAFT

Contents

01

The privacy you cannot audit, and the speed you cannot trust

Every cloud runs models on other people's data. Every vendor promises the data is safe. And nobody — not the clouds, not the confidential-computing marketing, not the compliance auditors — can show you the one fact that would settle it: **at the instant the model computed your answer, could anyone on that machine have read your input?**

What private inference is, and what gets measured instead

A privacy promise is a claim about a moment you never witness. Four properties decide whether the promise is real, and today a buyer can verify none of them:

Property	What the buyer needs	Verifiable today?
Confidentiality	Was the input ever in plaintext on the server — in a register, a cache, a core?	No. You are trusting a policy, a data-processing agreement, and a logo.
Correctness	Does the private computation return the same answer the plaintext model would?	Self-attested. The party doing the hiding also reports the result.
Latency	What did the privacy cost in wall-clock time, and could it be lower?	Quoted, never independently comparable, never competed down.
No trusted hardware	Does safety rest on a chip vendor's enclave staying unbroken?	Usually not — and enclaves break (see below).

The measurement gap has a structural cause, not an accidental one: **a privacy claim cannot be trusted when the claimant holds the plaintext.** A confidential-computing service decrypts your data inside an enclave to compute on it; a "we don't log" API sees every byte in the clear for a microsecond and asks you to believe the microsecond was harmless. Any evaluation in which the server ever possesses the plaintext is measuring a promise, not a property.

Fully homomorphic encryption (FHE) removes the moment entirely. The server computes directly on ciphertext and returns ciphertext; the input is never decrypted anywhere in the compute path, and only the key-holder can read the result. This is the one architecture where confidentiality is a mathematical property of the computation rather than a property of the operator's conduct.

The second problem: FHE is slow, and how slow is a choice

FHE's cost is famous — three to six orders of magnitude slower than plaintext. What is less appreciated is that **which end of that range you land on is not fixed by the model**. It is decided by how the model is compiled into a circuit: the quantization bit-widths, the acceptable per-operation error, the rounding strategy, whether a step needs an expensive bootstrap. The same trained model, compiled two ways, can differ by more than an order of magnitude in cost at the same accuracy. We measured a small neural net drop 2.4× in latency at equal accuracy from changing a single compiler dial.

That optimization surface is real engineering work, it compounds, and — crucially — **nobody is being paid to mine it competitively**. A vendor compiles once, ships it, and moves on. The frontier of "fastest correct encrypted circuit for model M" is left on the table.

There is a trap in paying for speed, though, and it is the reason a naïve latency market would be worse than nothing: **you can always make an FHE circuit faster by making it less secure**. Shrink the lattice dimension and the ciphertexts get smaller and the arithmetic gets cheaper — and the encryption

gets breakable. A market that rewards speed without pinning security doesn't fund faster privacy; it funds fake privacy that happens to benchmark well. So the missing primitive is precise: a competition to make encrypted inference faster, in which correctness is checked by a party that holds the answer key, security is fixed by the referee and cannot be bargained away, and the ranking cannot be gamed by whoever reports the benchmark. Cifra is that primitive, plus the incentive market built on it, plus a product built on the winner.

What changed

Three ingredients made this buildable now. First, FHE compilers crossed the usability threshold: Zama's Concrete-ML turns a trained scikit-learn or PyTorch model into an encrypted-inference circuit with a single call, and — because it targets the exact-arithmetic TFHE family — a correctly evaluated ciphertext decrypts to the same answer the plaintext model gives, bit for bit. Second, the same compiler exposes a **deterministic, hardware-neutral cost for any circuit** — its complexity — computed at compile time and identical on every machine that shares the pinned toolchain. That number is what lets every eligibility gate in a decentralized competition be exact: every honest validator computes the same pass/fail verdict for the same recipe, to the digit. The competitive ranking itself is by *measured* latency — the thing the buyer actually pays for — and it stays consensus-safe because validators only need to agree on the *ordering*, not on raw milliseconds (§4). Third, Bittensor's consensus supplies trustless aggregation, and its security level is a compiler input with a hard 128-bit floor — so a referee can guarantee the crypto rather than audit it.

> The frontier of fast encrypted inference is unmined because it has never had a referee — one that can rank two circuits by speed without trusting either author's benchmark, reject the one that quietly weakened the crypto, and confirm both still compute the published model. Cifra is that referee, and the prize money.

02

The contest is the compiler

The mechanism

An owner in Cifra does not publish a circuit. It publishes a **model** — a plaintext reference the network must learn to compute under encryption — together with the rules of the contest: the input distribution, an accuracy floor, a complexity band, and the exact toolchain version. Every miner then compiles its own circuit for that model and competes on how fast it can make it while still matching the reference. The loop, per miner, is:

```
OWNER   publishes   reference model (plaintext) + calibration inputs + rules
                    → one HuggingFace task repo; hashes pinned in the taskspec
                    rules = { accuracy_floor, complexity band [min, cap], p_error_floor, toolchain }

MINER   compile     its own FHE circuit for the model — choosing bit-widths, p_error,
                    rounding, distillation — to minimise encrypted latency above the floor
        recipe      emit recipe.json = { model: <concrete-ml JSON>, compile: { p_error } }
                    (a fitted-but-UNcompiled model + compile knobs — NOT a binary, NOT a pickle)
        publish     upload recipe.json to HuggingFace  →  set_commitment({repo, rev}) on-chain

VALIDATOR   for each miner's committed {repo, rev}:
        download    recipe.json at the pinned revision (anonymous, byte-capped)
        compile     the validator — never the miner — builds the compile Configuration:
                      security_level = 128  (enforced),   p_error = max(recipe, floor)
                    circuit = model.compile(calibration, …)        ← in a sandbox
        gate        accuracy(circuit vs reference) ≥ floor?   min ≤ complexity ≤ cap?   128-bit?
        measure     run real encrypted inferences (keygen excluded) → latency_ms
        score       rank eligible miners by latency (3% tie-band; earlier commit wins a tie)
        settle      copy-priority → geometric weights 1, 1/3, 1/9, … → set_weights
```

The ranked quantity is **measured latency**: the mean wall-clock milliseconds per real encrypted inference (`fhe="execute"`), with the one-time key generation excluded, read by each validator off its own hardware. Latency is what the owner ultimately buys, so it is what the market pays for. It is also a local measurement — two validators' clocks never agree to the digit — so the design keeps consensus tight in three ways. Only the *ordering* matters: weights are geometric down the leaderboard, so validators need to agree on ranks, not milliseconds. A **3% tie-band** absorbs timing jitter: same-machine repeats vary ~1–2% while real circuit improvements shift latency by tens of percent, and within a band the earlier on-chain committer — a block number every validator reads identically from the chain — ranks first. And every *eligibility* gate stays fully deterministic, built on `circuit.complexity` — the compiler's hardware-neutral cost, identical on every machine under the pinned toolchain (we verify this across separate processes as a build gate) — and on a task-seeded input set. Complexity is still measured and logged next to latency as the deterministic cross-check on every verdict.

Why the validator compiles, not the miner

This single design choice — miners submit a recipe, and the validator compiles it — buys two guarantees that a "miner ships a compiled circuit" design cannot have:

- **No arbitrary code on the referee.** A compiled FHE circuit is a native binary; loading a stranger's binary is native-code execution. A recipe is a JSON description of a fitted model, loaded through Concrete-ML's structured deserializer (`loaders.loads`), never `pickle`/`joblib`. There is no code in it to

run. Compilation still happens inside a sandboxed subprocess as defense in depth — no network egress and a read-only filesystem via bubblewrap, resource- and time-bounded, credentials stripped — but the attack surface starts far smaller.

- **The crypto is the referee's, not the contestant's.** Because the validator builds the compile `Configuration`, the security level is set by the party doing the grading. The miner supplies a model and one error knob; it never touches the parameters that decide whether the encryption is strong. Which brings us to the property the whole speed market rests on.

You cannot buy speed with weakness

The danger in any latency competition on FHE is that security and speed pull the same lever. An FHE ciphertext hides its message behind a random vector plus small noise; recovering the secret is a hard lattice problem (LWE), and its hardness is set by two knobs baked into the keys — the lattice dimension and the noise. Turn them down and the circuit gets smaller, cheaper, and faster — and breakable. A miner optimizing for low latency would love to hand you an 80-bit circuit.

Cifra makes that impossible by construction, not by policing:

1. **The validator sets the security level.** It compiles every recipe with `security_level = SECURITY_128_BITS`. The miner's submission cannot override it; the validator ignores any configuration the miner might try to smuggle in and builds its own.
2. **There is no weaker setting to ask for.** Concrete's `SecurityLevel` enumeration contains only `SECURITY_128_BITS` and `SECURITY_132_BITS` — there is no 80-bit or 100-bit rung in the API at all. Even a validator that wanted to grade a weak circuit has nothing below 128 to select.
3. **The scorer double-checks.** The evaluator reports the achieved security level with every verdict, and a reading below 128 zeroes the score regardless.

128-bit security means the best known attack needs about 2^{128} operations — the bar AES-128 and modern TLS are held to, and because it rests on lattice problems, it is post-quantum: a large quantum computer does not shortcut it. Because that bar is guaranteed for every submission, miners can only compete on legitimate optimization — better quantization, tighter error budgets, smarter rounding — never by cutting the cryptography. It is the property that lets a speed market run on a privacy product without the two goals fighting.

What the accuracy gate does, and does not, measure

The gate is what makes the miner's circuit *the owner's function* rather than some cheaper look-alike. The validator draws a fixed set of inputs from the task's published distribution — **deterministically seeded from the task id**, so the miner's own self-check and every validator draw the identical set — runs the miner's circuit and the reference on them, and requires their argmax outputs to agree at or above the floor. Because a correctly evaluated TFHE circuit is exact, the fast clear-quantized evaluation the gate uses equals what the encrypted circuit computes; the real encrypted executions that time the circuit for the leaderboard double as a spot-check that the deployed circuit runs and agrees. A circuit that over-quantizes itself into cheapness stops matching the reference and fails the gate. And for tasks whose reference is a large nonlinear model, the complexity band has a lower edge too: a toy circuit from a much smaller model class (a ~10^3-complexity logistic regression posing on a neural-net task) is excluded outright by `min_complexity` rather than argued with at the accuracy margin.

> **LIMITATION · FIDELITY, NOT DISTRIBUTION SHIFT** The gate measures agreement with the reference *on the task's declared input distribution*. It proves a circuit computes the published model there; it does not measure how the reference behaves on a specific customer's real data. Matching a model to a real distribution is the owner's job when defining the task, and the product layer's job on real (still client-encrypted) traffic. The chain never claims what it did not test.

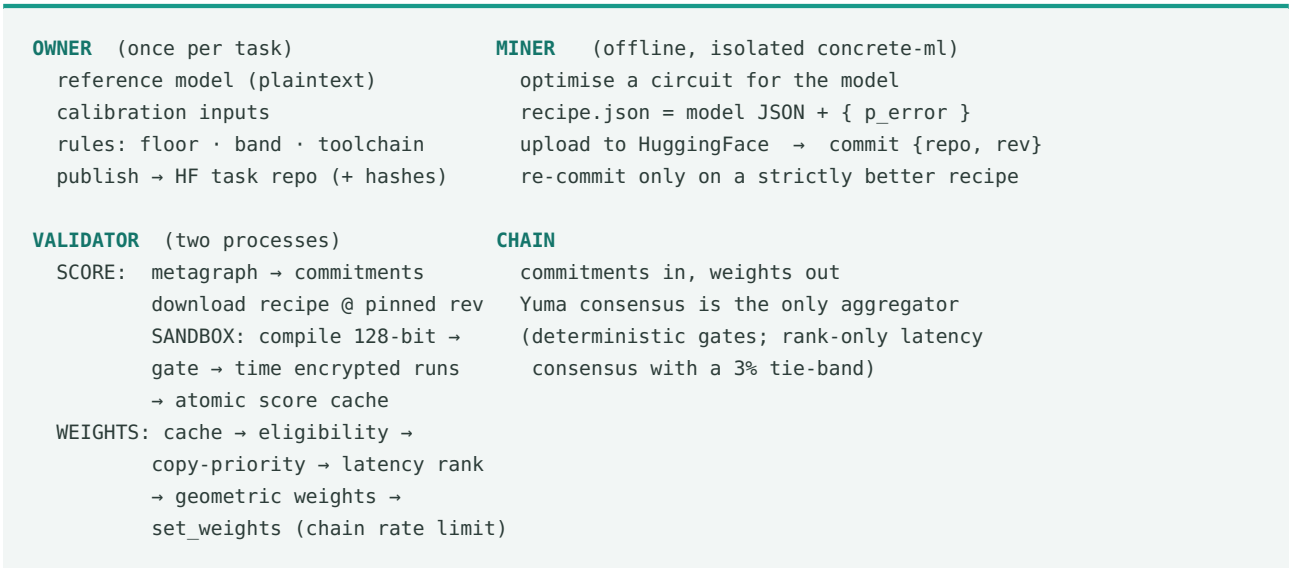
LIMITATION · MODEL SIZE FHE's overhead is real and model-shaped. Cifra v1 ships the tier that is honest today — encrypted tabular models whose circuits compile and grade in seconds on a laptop CPU. Small quantized neural nets are the next tier; deep CNNs and encoder transformers belong to a GPU tier on the roadmap; encrypted LLM inference exists in the literature at minutes per token and is a showcase, not an economic tier. The task manifest carries the reference model and toolchain, so difficulty rises by publishing a new task, never by changing the protocol.

03
The network

Nothing central, by construction

The design has no owner-operated organ in the live loop — no key server, no challenge feed, no serving endpoint, no results API. There is nothing to serve and nothing to query:

- **Tasks are published artifacts.** The owner writes a reference model, a calibration set, and a rules file once, hashes them, and publishes all three to a HuggingFace task repo whose id is the only thing miners and validators need; after that the owner is out of the loop until it chooses to publish a new task. Miners and validators watch the repo head and re-sync themselves when a new task appears. The owner sets no weights and grades nothing.
- **Miners hold nothing of the validator's.** A miner never receives a key, a ciphertext, or a query. It optimizes offline, publishes a recipe to HuggingFace, and writes a pointer to the chain. It runs no server and answers no traffic.
- **Grading is validator-local.** Each validator downloads the same public recipes, compiles them with its own trusted toolchain under its own 128-bit configuration, gates them on checks it computes itself, and times encrypted inference on its own hardware. No score is submitted anywhere. (A validator *may* publish its raw scoring state to a HuggingFace repo for dashboards — but that is observability, not consensus; nothing reads it in the weight path.)
- **Aggregation is the chain itself.** Yuma's stake-weighted consensus is the only place independent judgments meet. Every eligibility verdict is deterministic — honest validators compute the same complexity and the same pass/fail for the same recipe, to the digit — and the ranking needs only ordinal agreement, which the tie-band protects against timing jitter. Consensus on who competes is exact; consensus on the ordering is designed to be stable rather than assumed to be identical (§5).



The miner lifecycle

1. **Optimize.** Load the owner's reference model and search the compile space — bit-widths, `p_error`, rounding, distillation — for the fastest circuit that still clears the accuracy floor and sits inside the complexity band. The miner self-evaluates with the exact same evaluator the validator uses, on the exact same task-seeded gate inputs, so "what I measure is what I am scored on" holds by construction for every deterministic gate — and its local latency ranking is the same measurement

the validator will make, on the validator's hardware. A reference logistic-regression optimizer ships in-repo as the baseline strategy to beat.

2. **Publish.** Emit `recipe.json` and upload it to a HuggingFace repo; capture the commit revision.
3. **Commit.** Write `{repo, rev}` to the chain with `set_commitment` under the miner's own hotkey. The revision is a full commit hash, so the pointer is immutable — a validator fetches exactly what the miner committed, or nothing. The chain records the commit block, and that block number matters: it is the tie-breaker that lets an incumbent keep its rank against a copy-and-tweak challenger (§4).
4. **Idle.** The miner is otherwise a quiet loop: re-optimize, and re-commit only when it finds a strictly better recipe, so the on-chain pointer is stable. No axon, no serving, no forward/blacklist/priority, no keys, no per-query work of any kind.

The validator: two loops, one cache

A validator runs as **two processes that share one persistent score cache** — a deliberate split, so a long compile never delays a weight update:

The **score process** is a plain loop: sync the task (noticing a newly published one), read the metagraph, and for each miner uid with a commitment, evaluate it — re-evaluating only when the miner's `(repo, rev)` or the task revision changes, so a stable circuit is graded once and cached:

1. **Read the pointer.** `get_commitment(uid)` → `{repo, rev, block}`; skip miners that have committed nothing.
2. **Fetch.** Download `recipe.json` at the pinned revision, anonymously, with a byte cap — a hostile repo cannot force an unbounded pull, and only that one file is ever fetched.
3. **Evaluate in the sandbox.** Hand the recipe to the isolated concrete-ml subprocess, which loads it without pickle, compiles it under the forced 128-bit configuration, runs the accuracy gate, executes and times a handful of real encrypted inferences (keygen excluded), and returns a small JSON verdict — complexity, bootstrap count, accuracy, achieved security, latency, wall-clock. A recipe that hangs, over-runs its CPU budget, OOMs, or crashes yields a failed verdict, never taking the validator down.
4. **Cache and publish.** Write the verdict to the on-disk cache (atomic writes), and optionally upload the full raw scoring state — every miner's verdict, eligibility, reject reason, rank, and implied weight — as one JSON file per task in a public HuggingFace results repo, for dashboards and auditability.

The **weights process** reloads that cache, filters to the current task, applies eligibility and copy-priority, ranks the survivors by latency with the tie-band, assigns geometric weights 1, 1/3, 1/9, … down the leaderboard, and calls `set_weights` whenever the chain's own `weights_rate_limit` allows. It is cheap, never blocked behind a compile, and re-derives everything from the cache on every pass.

The two runtimes — a bittensor chain process and a concrete-ml evaluator — are separate for a second reason: the two stacks pin mutually incompatible numeric libraries, so the FHE runtime must live in its own environment, and that isolation doubles as the security boundary. The chain process, which holds the wallet, never imports concrete-ml and never runs a line of miner-influenced code — it only ever reads a small verdict back from the sandboxed subprocess. The one rule that keeps both properties true: **no module imports both `bittensor` and `concrete-ml`.**

04
The score

Gates, then a leaderboard

Hard gates run first; any failure records its reason and zeroes the miner for the round. Every gate is deterministic — identical on every honest validator under the pinned toolchain:

Gate	Rule
Ran	the sandbox compiled the recipe, the accuracy run completed, and the timed encrypted executions agreed with the clear-quantized output; a crash, timeout, or over-budget compile is a zero, not a wrong answer
Security	the achieved security level is ≥ 128 bits — structurally guaranteed by the forced configuration, and re-checked here
Complexity band	`min_complexity ≤ circuit.complexity ≤ max_complexity`. The cap (default 4 × `reference_complexity`) is a guardrail so a pathological circuit cannot grief the evaluator; the floor (default 0, set per task) excludes toy model classes outright — a ~10^3-complexity logistic regression cannot pose on a neural-net task
Accuracy	argmax agreement with the reference on the task-seeded gate set is ≥ `accuracy_floor`
Toolchain	the validator's `concrete-ml`/`concrete-python` versions match the task's pin; a mismatched validator excludes itself from scoring rather than emit divergent numbers

Then the survivors are ranked. There is only one competitive axis, and it is the one the buyer pays for:

```
RANK    order eligible miners by latency_ms — the mean measured milliseconds per
        real encrypted inference (fhe="execute"), keygen excluded, lowest first

TIE     latencies within 3% of the band leader's count as EQUAL; inside a band,
        the earlier on-chain committer ranks first (then hotkey, for a total order).
        Bands are leader-anchored — each member is compared to the band's fastest
        entry, not its neighbour, so near-ties cannot chain into one giant band.

PAY     weight(rank i) = (1/3)^i  →  1, 1/3, 1/9, …   normalised on-chain
```

Three properties make this the right curve for a locally measured metric. First, **consensus needs only the ordering**: geometric weights depend on ranks alone, so two honest validators agree on payouts whenever they agree on who is faster than whom — never on raw milliseconds. Second, **the tie-band absorbs exactly the noise that exists**: repeat timings on one machine jitter ~1–2%, while a real circuit improvement moves latency by tens of percent; a 3% leader-anchored band means measurement noise cannot reorder near-equals, and the deterministic tie-breaker — the on-chain commit block, which every validator reads identically — settles them. Third, **the ratio is steep but not winner-take-all**: the leader earns 3× the runner-up, so genuine speed is paid steeply, but a funded pipeline of challengers survives behind the champion rather than one fragile winner taking everything.

The tie-band has a second job: it is an anti-leapfrog device. A challenger must be *genuinely more than 3% faster* to displace an incumbent — a copy of the leader's public recipe with a cosmetic tweak lands inside the band and ranks *behind* the original on commit priority. Re-registering a hotkey resets the commit block too, so churning identities forfeits tie priority rather than gaming it.

Copy-priority: the earliest committer owns the recipe

Because recipes are public on HuggingFace, a lazy miner could copy a rival's `recipe.json` and commit it as its own. Cifra fingerprints each recipe (a SHA-256 over its canonical envelope) and tracks the earliest chain block at which each fingerprint appeared. If a miner's fingerprint first appeared earlier under a different hotkey, that miner scores zero — the original committer keeps the reward. Copying a winner is never profitable; you have to build a faster one.

From scores to weights

```
eligible → copy-priority → latency rank (3% tie-band) → w[uid] = (1/3)^rank
weights normalised and set on-chain whenever weights_rate_limit allows
a re-committed recipe is re-evaluated; an unchanged one is served from the cache
a miner whose evaluation failed is simply absent from the board this round
```

There is deliberately no smoothing constant to tune: the leaderboard *is* the weight vector, recomputed from cached verdicts on every pass and refreshed the moment a re-evaluated recipe changes the ordering. A better circuit shipped today earns today. Stability comes from the mechanism itself rather than from averaging: verdicts are cached per `(hotkey, revision, task)` so an unchanged recipe cannot wobble, the tie-band keeps timing jitter from reordering near-equals, and commit priority keeps the board stable under churn.

Measured behaviors: the shipped demo

The repository ships an end-to-end demo — real Concrete-ML on both sides, no chain, no accounts — that builds a task, has a miner optimize against it, and grades the result exactly as a validator would. The shipped task is real: `breast-cancer-v1`, a diagnostic classifier over 30 standardized tumor measurements (`reference_complexity = 1315`, `accuracy_floor = 0.90`). The miner's candidate sweep produces (latencies measured on the development machine — validator-local by design):

Circuit	Complexity	Accuracy vs reference	Latency	Verdict
Owner baseline (8-bit)	1315	—	—	the reference compile — the function to match
Over-quantized (3-bit)	673	0.881	—	**rejected** — below the 0.90 floor
Winner (4-bit)	**788**	**0.935**	**14.6 ms**	**rank #1** — fastest eligible circuit
Faithful (6-bit)	1024	0.951	15.0 ms	passes; within the 3% tie-band of the leader — the earlier committer of the two keeps the band's top rank
Conservative (8-bit)	1259	0.957	15.2 ms	passes; outside the leader's band → next rank
Weak-crypto-for-speed	—	—	—	**impossible** — no sub-128 setting exists to request
Plagiarized winner (later)	788	0.935	14.6 ms	**scores 0** — fingerprint committed earlier elsewhere

The sweep is the design in one line: the miner quantizes exactly as aggressively as the gate allows — the 4-bit circuit lands above the fidelity floor at 40% less circuit cost than the owner's own compile and the

fastest measured latency — while the 3-bit circuit that went one step too far is rejected, the weak-crypto shortcut has no rung to stand on, and a copyist earns nothing. The sweep is also honest about the low end of the difficulty range: this task's circuits are linear (zero bootstraps), so the measured latencies cluster within a few percent and the tie-band, not the stopwatch, does the ranking work between them — with commit priority rewarding whoever published first. Latency separates decisively on the bootstrap-bearing neural-net tiers, where a compiler dial moves it by tens of percent (§1); the tabular tier's job is to prove the pipeline end to end.

05
Threat model

Attacks and their answers

The design assumes every miner is rational and adversarial. Attacks, and where each one dies:

Attack	Answer
Sell speed by weakening the crypto	Structurally impossible: the validator builds the 128-bit configuration, and Concrete's security enum has no sub-128 rung to request. The verdict re-checks the achieved level and zeroes anything below 128.
Ship a fast circuit that no longer computes the model	The accuracy gate runs the circuit against the reference on the task's own distribution and rejects it below the floor; a cheaper look-alike that disagrees fails. On nonlinear tasks the complexity floor additionally rejects the whole toy-model class before accuracy is even argued.
Ship a native binary that runs code on the validator	There is no binary. A recipe is JSON loaded through a structured deserializer (never `pickle`/`joblib`), and compiled in a sandboxed subprocess: no network egress and a read-only filesystem (bubblewrap, with an `unshare` network-isolation fallback), CPU- and wall-clock-bounded, credentials stripped.
Report a fake, low latency or complexity number	The miner reports nothing. Every validator recompiles the recipe itself, reads complexity off its own circuit, and times encrypted execution on its own hardware; no miner-supplied measurement exists in the pipeline.
Copy a rival's published recipe	Recipes are fingerprinted; the earliest on-chain committer of a fingerprint keeps the reward, later copies score zero.
Copy-and-tweak to dodge the fingerprint	A perturbed copy is a new fingerprint — but unless it is genuinely >3% faster it lands inside the incumbent's tie-band and ranks behind it on commit priority. Leapfrogging by cosmetic edits pays nothing; only real speed moves rank.
Grief the validator with a huge or hostile recipe	The download is byte-capped and single-file; the compile runs under a wall-clock timeout and CPU rlimit inside the no-egress sandbox; a hang or OOM yields a failed verdict, not a crashed validator. The complexity cap bounds what an eligible circuit may cost to grade.
Register many hotkeys off one backend	Each hotkey still needs a genuinely faster circuit to rank; duplicates collapse under copy-priority, geometric weights pay one faster circuit more than a spread of identical ones, and re-registration resets commit priority.
Game the ranking with faster hardware	Miner hardware cannot move the score at all: the validator measures on its own machine, and the deterministic gates are hardware-neutral by construction. Validator hardware affects only its own local timings — see the residual below for why the rank-only design keeps that from forking consensus.
Copy another validator's weights	Gates are deterministic, so an honest validator reproduces the same eligible set anyway; the ranking is re-derived locally from its own measurements. Copying reveals nothing and gains nothing, while off-consensus weight vectors are penalized by Yuma's stake-weighted clipping.

What we do not claim

RESIDUAL · LATENCY IS A LOCAL MEASUREMENT The competitive metric is wall-clock latency on the validator's own hardware, and two validators on very different machines can legitimately disagree on *adjacent* ranks. The design contains this — only the ordering enters consensus, the 3% tie-band absorbs same-machine jitter and hands near-ties to a chain-read tie-breaker, and Yuma's stake-weighted median damps outliers — but it does not eliminate it. Keeping validator hardware roughly uniform keeps adjacent ranks stable, and that is an operational recommendation, not a protocol guarantee. The deterministic complexity number is logged beside every latency as the hardware-neutral cross-check.

RESIDUAL · NEAR-COPIES Fingerprint dedup catches an exact copy; the tie-band makes a cosmetic near-copy unprofitable (it ranks behind the original unless >3% faster). What remains open is a rival who takes the *idea* — retrains one bit-width, restructures the quantization — and lands a real improvement; industry-wide, nobody has solved near-copy detection, and we say so. The geometric (non-winner-take-all) curve blunts the payoff, and behavioral fingerprinting — correlating rare shared quirks across recipes — is a planned detector that will log first and penalize only after calibration on live data.

RESIDUAL · TOOLCHAIN AS TRUST ANCHOR Deterministic gate verdicts hold only within one pinned `concrete-ml`/`concrete-python` build; a version drift changes complexity numbers and would fork the eligible set. Cifra pins both versions in the task and makes a mismatched validator exclude itself, so a drifting validator goes quiet rather than divergent — but the guarantee is only as strong as the pin's discipline. Reproducible, independently re-derivable toolchain builds are the structural close, and a named delivery phase.

RESIDUAL · SANDBOX DEPTH The evaluator today runs as a separate process under the strongest isolation the host supports — bubblewrap with no network, a read-only filesystem, and a private `/tmp` (falling back to `unshare` network isolation, then to a plain bounded subprocess with a logged warning) — plus a wall-clock timeout, a CPU rlimit, and a credential-stripped environment, backed by the fact that a recipe carries no code. It is not a VM: a kernel or compiler exploit is out of scope of what the jail can contain, and hosts without bubblewrap degrade to weaker isolation rather than refusing to run.

BOUNDARY · WHAT THE CHAIN PROVES On-chain, Cifra proves that a miner's circuit computes the published reference on the task's declared distribution at 128-bit security, within the task's complexity band, under the pinned toolchain — and pays by each validator's measured speed ordering of those circuits. Nothing more. Real-world quality on a specific customer's data is measured off-chain by the product layer, on real client-encrypted traffic, and the chain never claims otherwise.

Reproducibility. Given a validator's evaluation record — the commitment, the recipe revision, the task-seeded gate inputs, and the pinned toolchain — every *gate* verdict recomputes exactly from public code and constants: the recipe is public, complexity is deterministic, and the gate inputs are derivable from the task id. The latency measurement re-runs to within timing noise on comparable hardware, and the published per-task results file (when a validator enables it) exposes every verdict, rank, and implied weight for anyone to audit. Appendix A walks one such record end to end.

06
Economics and the product

Two ledgers

The subnet and the company sit on deliberately separate ledgers. The subnet is self-sufficient on emissions: miners and validators are paid in the subnet's alpha token for producing the one thing the chain produces trustlessly — a continuously refreshed leaderboard of the fastest correct, provably 128-bit-secure circuits for the owner's models — whether or not anyone ever buys anything. The company is revenue-driven and out of the weight path: it consumes public chain state like any third party could, and its only on-chain influence is exercised ahead of time, in the shipped task definitions and protocol code.

The winner is a deployable artifact

This is where the model-competition design pays off over a serving competition. In a subnet where miners serve encrypted inference, the "winner" is a fast operator you must keep querying. In Cifra, the winner is a circuit — an optimized, 128-bit, accuracy-verified `recipe.json` that anyone can compile and run. The operator takes the current top recipe and promotes it into its own hosted, key-managed serving stack. A customer encrypts locally, sends ciphertext, and receives ciphertext only it can read; the operator runs the promoted circuit and, by construction, cannot see the input, the output, or the result. Confidentiality is not a policy the customer audits — it is a property of the math, and it survives even a fully compromised server.

That is the strongest thing FHE has over its alternatives, and it is worth stating against the incumbent. Confidential-computing enclaves decrypt data inside a chip and rest on that chip staying unbroken; in October 2025 the TEE.fail and WireTap results extracted enclave secrets with roughly a thousand dollars of interposer hardware. FHE has no such moment — there is no enclave to break because the data is never decrypted on the server at all. The subnet is the product's R&D engine and hiring market: a permanent, adversarial, continuously refreshed competition to make that private computation faster, feeding a normal SaaS with an abnormal supply chain.

> **SCOPE · THE PRODUCT LAYER IS NOT PART OF V1** Everything in this section describes where the winning artifact can go, not code that ships in the subnet repository. The deliverable today is purely the competition: miners commit optimized circuits, validators grade and rank them, weights reflect the fastest correct one. What consumes the winner — the key-managed serving stack, enterprise workflow, billing — is a separate concern, deliberately out of the current build.

Revenue, tokens, positioning

> **ILLUSTRATIVE, NOT COMMITTED** Everything in this subsection is proposed design. There is no user revenue at launch, no token sale, and no promise of token value. The alpha token is Bittensor's standard emission mechanism, not a fundraising instrument.

Line	Buyer	Shape (illustrative)
Private-inference SaaS — encrypted scoring for regulated data (risk, fraud, eligibility, triage)	Health, finance, and public-sector teams barred from sending plaintext to a cloud	Per-seat or per-model subscription
Encrypted-inference API		Per encrypted-query pricing

Line	Buyer	Shape (illustrative)
	App builders needing "compute on my user's data without seeing it"	
Circuit certification	Vendors and auditors vetting an encrypted-inference stack	Per-report — selling the measurement itself

Positioning. Against confidential computing, Cifra does not compete on speed — it removes the trust assumption on the one axis (no enclave to break) enclaves cannot answer. Against secure multi-party computation, it is non-interactive and single-server, with no non-collusion assumption and no gigabytes of per-query traffic. Against a plaintext cloud API, it is the difference between a data-processing agreement and a mathematical impossibility. A verified, speed-ranked, 128-bit-guaranteed market for encrypted circuits with no trusted hardware is an unoccupied corner.

Tokens. Standard per-subnet emissions pay miners, validators and their stakers, and the owner. Under dTAO the emission pool scales with TAO staked into the subnet's alpha token: conviction deepens rewards, rewards attract miners, miners push the frontier of fast correct circuits, the product gets better, and the loop closes — while the ranking itself stays purchasable only by building a faster, correct, secure circuit, because weights follow measurements no amount of stake can fake.

07
Status and delivery

What exists today

Cifra was rebuilt from a serving design into this model-competition design after a measured finding: with the owner publishing a fixed circuit, miner latency differs only by hardware and network — a commodity race the network can never win against the owner's own compiler. Recompiling one model with a single different dial gave 2.4× lower latency at equal accuracy; that entire competitive surface was being wasted. The scoring then moved once more, from paying a complexity curve to paying the latency *ranking* — complexity is the perfect referee's ruler but a proxy, and the moment real encrypted executions were already running for the correctness spot-check, timing them and ranking on the measurement (with the tie-band making the local measurement consensus-safe) paid miners for the thing itself. The current codebase is a clean, purpose-built package — no serving, no data plane, no template scaffolding. As of this draft:

	Component
Built and tested	Latency-ranked scoring — deterministic gates (128-bit, accuracy, complexity band), leader-anchored 3% tie-band with commit-block priority, geometric weights, copy-priority · recipe envelope with JSON-only (no-pickle) load + canonical fingerprint · task-seeded accuracy sampling · validator-side compile forcing 128-bit + p_error floor · sandboxed evaluator (bubblewrap: no network, read-only filesystem, private /tmp; unshare fallback; CPU + wall-clock bounded, credentials stripped) returning a JSON verdict with measured encrypted latency · two-process validator (score loop + weight loop) sharing an atomic on-disk cache, re-evaluating only changed commitments · toolchain-pin self-exclusion · chain client (metagraph, get/set_commitment with commit block, set_weights under the chain's rate limit) · HuggingFace task publishing (owner tool, separate repo), rev-pinned byte-capped recipe download, and optional per-task results publishing for dashboards · miner / validator roles and a single CLI · a real task — `breast-cancer-v1`, a diagnostic classifier on 30 standardized tumor features — plus a real-Concrete-ML competition demo and a full validator round against a fake chain; unit, real-FHE, and validator-loop tests passing; lint and type checks clean
Written, not yet run on a live chain	The end-to-end on-chain loop — miner `set_commitment` → validator `get_commitment` → download → sandbox-evaluate → rank → `set_weights` — is implemented against the documented SDK and exercised against a fake chain with real FHE compilation; it has not yet run on a real subtensor node (local-chain bring-up needs Docker, unavailable in the build environment)
Not yet done	Subnet registration and testnet soak · commit-reveal weights · near-copy behavioral fingerprinting · larger model tiers (small NN, then GPU CNN/transformer) · reproducible toolchain builds · everything in the delivery plan below

The shipped task compiles a reference at complexity 1315 under 128-bit; the demo miner's winning 4-bit circuit passes the 0.90 gate at complexity 788 (40% cheaper) and the fastest measured encrypted latency of the sweep, and a second validator process recomputes the identical 788 — the gate determinism the eligible set rests on, verified rather than assumed.

Delivery plan

Phase	Scope	Status
D0		done

Phase	Scope	Status
	Reference implementation: gates + latency-rank scoring, FHE compile/evaluate/time, bubblewrap sandbox, chain/HF clients, owner task publishing, roles, CLI, competition + validator-loop tests, demo	
D1	Live chain: local-subtensor and testnet dry-run of the full commit → evaluate → rank → set_weights loop (honest vs over-quantized vs plagiarist vs copy-and-tweak vs weak-crypto); register; recruit miners; observe cross-validator rank stability on heterogeneous hardware	next
D2	Commit-reveal weights; near-copy behavioral-fingerprint detection (log → calibrate → enforce); validator hardware guidance from D1 latency-spread data	planned
D3	Reproducible toolchain builds anyone can re-derive to the pinned versions — turning the toolchain trust anchor into a checkable property	planned
D4	Small quantized neural-net task tier (the first task needing bootstrapping — where measured latency separates by tens of percent and the complexity floor earns its keep); larger calibration and complexity budgets	planned
D5	GPU tier: encrypted CNN and small encoder-transformer tasks, added as new task manifests with no protocol change	planned
—	Product layer (separate repository): key-managed serving that promotes and hosts the winning circuit	planned

Risk register

Risk	Failure it threatens	Standing answer
FHE economics	Optimizing circuits costs more than emissions return → miners leave	Launch on the cheap tabular tier where honest optimization is coverable; raise difficulty only by publishing harder tasks as hardware and emissions rise; watch miner churn as a health signal. This, not cryptography, is what ends FHE-subnet attempts.
Weak crypto for speed	A speed market funds fake privacy	Structurally impossible: the validator owns the 128-bit configuration and the API has no weaker rung; the verdict re-checks. The single most important property, and it is free.
Latency divergence	Validators on unlike hardware rank adjacent miners differently → noisy weights	Rank-only consensus, the 3% tie-band with a chain-read tie-breaker, and Yuma's stake-weighted median contain it; complexity is logged as the deterministic cross-check; D1 measures the real cross-validator spread and D2 turns it into hardware guidance.
Toolchain drift	Validators compute divergent gate verdicts → forked eligible set	Both versions pinned in the task; a mismatched validator self-excludes; reproducible builds (D3) make the pin checkable rather than trusted.
Recipe as attack vector	A hostile submission runs code or griefs the referee	JSON-only structured load (no pickle), byte-capped fetch, bubblewrap sandbox (no network, read-only filesystem), CPU + time bounds.
Near-copies	A perturbed copy defeats fingerprint dedup	The tie-band makes a <3%-faster tweak rank behind the original; geometric (non-winner-take-all) rewards blunt the payoff; behavioral fingerprinting planned at D2.

Risk	Failure it threatens	Standing answer
Synthetic-to-real gap	A winner faithful to the task underwhelms on real data	The chain claims fidelity on the task distribution only; real-data quality is a product-layer measurement on real client-encrypted usage.
Slot competition	A subnet without miners, stake, and usage loses its slot	A real product wedge (regulated private inference is a paid market today), emissions-funded runway, and a launch scope small enough to be excellent at.

08

The claim we want to be held to

Private inference does not lack cryptography; it lacks a referee, and the fast frontier of it lacks a market. Fully homomorphic encryption has been real for years and improves monthly, yet the question a buyer actually cares about — what is the fastest circuit that computes this model correctly without weakening the encryption? — has never been competed down by anyone who could prove the answer. Cifra's contribution is that referee and that market: an owner publishes a model, miners race to compile the fastest faithful circuit for it, and validators admit them through gates every one of them computes to the same digit — then pay them by the measured speed of real encrypted inference, on circuits whose security the validators set themselves and cannot bargain away.

We prefer falsifiable commitments to adjectives, so we end with the claims this document should be held to. The chain proves fidelity-to-reference on the task's distribution, at 128-bit security, within a deterministic complexity band, under a pinned toolchain — and pays by a latency ordering each validator measures itself, made consensus-safe by rank-only weights and the tie-band — and nothing more, until the product layer measures real usage. Security-for-speed is impossible by construction, not by policing, for as long as the validator builds the configuration and the API offers no weaker setting — both true today. Latency is a local measurement and the paper says so: ordering, not milliseconds, is the consensus object, and adjacent-rank stability on unlike hardware is a thing D1 measures rather than a thing we assert. The toolchain pin is a trust anchor held by discipline until reproducible builds make it checkable. Copy-priority defeats exact copies, the tie-band defeats cosmetic near-copies, and genuine near-copy detection remains open — the paper says so. The FHE cost curve confines v1 to small models, and the paper says which. Every one of these boundaries has a named phase in the delivery plan — and the code that closes each one will be public, like everything else in Cifra.

Appendix A
One recipe, end to end

One task, one miner, one validator — the exact path the shipped demo exercises. Every number is the demo's real measured output on the shipped `breast-cancer-v1` task (latencies from the development machine — validator-local by design); every formula and constant is the shipped default.

What the owner published

Field	Value
`task_id`	`breast-cancer-v1`
reference model	a plaintext 8-bit logistic regression over 30 standardized tumor measurements, trained on the real Wisconsin diagnostic dataset (malignant vs benign), published as Concrete-ML JSON + its SHA-256
calibration	128 representative standardized patient inputs + SHA-256; the gate samples the matching `normal` distribution
`accuracy_floor`	0.90 argmax agreement with the reference on 1000 task-seeded inputs
complexity band	`min_complexity` 0 · `max_complexity` 5260 (= 4 × the reference's own compile)
`p_error_floor`	2^{-8}
`reference_complexity`	1315 — the owner's plain 8-bit compile
toolchain	`concrete-ml 1.9.0` · `concrete-python 2.10.0`, pinned
published as	three fixed-name files — `reference.json`, `calibration.npy`, `taskspec.json` — in one HuggingFace task repo; miners and validators need only the repo id

What the miner did. It loaded the reference, built a distillation set from the task's distribution, and swept quantization bit-widths, self-checking each candidate with the same evaluator and the same task-seeded gate the validator will use:

Candidate	Complexity	Accuracy	Latency	Passes gate?
2-bit	591	0.775	—	no — far below the 0.90 floor
3-bit	673	0.881	—	no — below the 0.90 floor
4-bit (chosen)	**788**	**0.935**	**14.6 ms**	**yes — fastest passer**
6-bit	1024	0.951	15.0 ms	yes, but not faster
8-bit	1259	0.957	15.2 ms	yes, but not faster

It emitted `recipe.json = { model: <4-bit model JSON>, compile: { p_error: 0.05 } }`, uploaded it to HuggingFace, and committed `{repo, rev}` on-chain — the chain recording the commit block.

What the validator did — its own record

Step	Action	Result
read	`get_commitment(uid)`	`{repo, rev, block}`
fetch	download `recipe.json` @ `rev`, byte-capped	4-bit model + p_error 0.05
compile	sandbox: `model.compile(calibration, p_error = max(0.05, 2⁻⁸), security_level = 128)`	circuit built
complexity	read off the circuit	788 — identical to the miner's self-check and to a second validator process; inside [0, 5260]
security	read achieved level	128-bit — forced; ≥ 128
accuracy	circuit vs reference on the task-seeded gate	0.935 ≥ 0.90 → pass
execute	3 real encrypted inferences, keygen excluded, outputs checked against the clear-quantized circuit	all agree; mean 14.6 ms → the ranking key

Scoring

```
gate    accuracy 0.935 ≥ 0.90,  security 128 ≥ 128,  0 ≤ 788 ≤ 5260      → all pass

rank    eligible latencies: 14.6ms (this miner), 15.0ms, 15.2ms
        band leader 14.6ms → band edge 14.6 × 1.03 = 15.04ms
          15.0ms  → INSIDE the band  → tied with the leader; earlier commit block first
          15.2ms  → outside the band → next rank

pay     weights (1/3)^rank → 1, 1/3, 1/9 → normalised: 0.692, 0.231, 0.077

a plagiarist committing this same recipe.json later, under another hotkey:
        fingerprint first seen earlier elsewhere → weight 0
a copy-and-tweak 1% faster: inside the incumbent's band → ranks behind it
a weak-crypto circuit at 80-bit for more speed:
        no such setting exists to request → cannot be built
```

Settlement. The weights process rebuilds this leaderboard from the score cache and sets the geometric vector on-chain whenever the chain's `weights_rate_limit` allows. A miner that finds a genuinely faster faithful circuit (>3% past the leader) re-commits and takes the top rank; one that copies a winner earns nothing; one that tweaks a winner cosmetically stays behind it; one that over-quantizes past the floor is rejected outright.

> **BOUNDARY, RESTATED** This record proves the 4-bit circuit computes the published reference on the task's distribution, at 128-bit security, at deterministic complexity 788, under the pinned toolchain — and that on this validator's hardware it was the fastest eligible circuit. Ordering is the consensus object; the milliseconds are local. Real-customer quality is a product-layer measurement on real client-encrypted usage (§5, §6).

Appendix B
Protocol constants

Consensus defaults as shipped. Scoring constants change only with a coordinated release; per-task rules travel in the task's own `taskspec.json`.

Constant	Default	Role
`LATENCY_TIE_BAND`	0.03	latencies within 3% of the band leader's count as equal; inside a band the earlier committer ranks first
`GEOMETRIC_RATIO`	1/3	each rank down the leaderboard earns this fraction of the rank above it (1, 1/3, 1/9, …)
`execute_spotcheck`	3	real encrypted inferences per verdict — the correctness spot-check and the latency sample (keygen excluded)
`accuracy_floor`	0.90 (shipped task)	min argmax agreement with the reference; set per task
`min_complexity`	0 (shipped task)	circuit-cost floor; set >0 on nonlinear tasks to exclude toy model classes
`max_complexity`	4 × `reference_complexity`	circuit-cost ceiling (guardrail)
`p_error_floor`	2^{-8}	minimum per-operation error the validator will compile with
`n_accuracy_samples`	1000	size of the deterministic, task-seeded accuracy-gate set
`security_level`	`SECURITY_128_BITS`	forced by the validator; the API offers no weaker rung
`eval_timeout_s`	300	per-recipe sandbox wall-clock budget (CPU rlimit scales with cores)
weights cadence	chain `weights_rate_limit`	`set_weights` fires as soon as the chain's own rate limit allows — no protocol-side epoch constant
`RECIPE_MAX_BYTES`	64 MB	download cap on `recipe.json`
`TASK_FILE_MAX_BYTES`	256 MB	download cap per task artifact (reference, calibration)
`concrete-ml` / `concrete-python`	1.9.0 / 2.10.0	pinned toolchain; validators must match or self-exclude

Glossary

FHE
Fully homomorphic encryption: computing directly on ciphertext so the input is never decrypted on the server; only the key-holder can read the result.

TFHE
The FHE family Cifra uses via Concrete-ML. It is exact — a correct evaluation decrypts to the plaintext model's output bit-for-bit — which is what lets the accuracy gate compare a circuit to the reference using fast clear evaluation.

Concrete-ML
Zama's compiler turning a trained scikit-learn/PyTorch model into an FHE inference circuit. A model is fitted, serialized to JSON (`dumps`) and reloaded (`loaders.loads`) without pickle, then `compile()`d into a circuit.

recipe
What a miner submits: `{ model: <Concrete-ML JSON of a fitted, uncompiled model>, compile: { p_error } }`. A description, not a binary; carries no executable code.

latency (`latency_ms`)
The competitive metric: mean measured milliseconds per real encrypted inference (`fhe="execute"`), keygen excluded, timed by each validator on its own hardware. Enters consensus only as an ordering.

tie-band
Latencies within 3% of their band leader's count as equal; inside a band the earlier on-chain committer ranks first. Absorbs timing jitter and makes cosmetic copy-and-tweak submissions rank behind the original.

geometric weights
The payout curve down the latency leaderboard: rank i earns $(1/3)^i$, normalized on-chain. Rank-only, so validators need agree only on ordering — and steep without being winner-take-all.

circuit complexity
The optimizer's deterministic, hardware-neutral cost estimate for a compiled circuit, read off `circuit.complexity`. Identical across machines under a pinned toolchain — the reason every eligibility gate is exact. Bounded per task by the complexity band, and logged beside latency as the cross-check.

programmable bootstrap (PBS)
The noise-reset step that lets an FHE circuit run to arbitrary depth and evaluate non-linear functions; the dominant cost of deep encrypted models. Reported per verdict.

p_error
The tolerated probability of a per-operation error in the compiled circuit. A miner may raise it for speed; the validator floors it so accuracy stays meaningful.

LWE / lattice dimension
Learning-With-Errors, the hard problem FHE security rests on. Its difficulty is set by the lattice dimension and noise; lowering them speeds the circuit and weakens the encryption — which is why the validator, not the miner, sets them.

128-bit security
The best known attack needs about 2^{128} operations; the bar AES-128 and modern TLS meet, and post-quantum because it rests on lattices. Forced on every Cifra circuit; the security API has no weaker option.

accuracy gate

The check that a miner's circuit computes the owner's model: argmax agreement with the reference on a deterministic, task-seeded input set, required at or above the task's floor.

copy-priority / recipe fingerprint
A SHA-256 over the canonical recipe envelope; the earliest on-chain committer of a fingerprint keeps the reward, later copies score zero.

toolchain pin
The exact `concrete-ml`/`concrete-python` versions in the task. Validators must match to keep gate verdicts identical; a mismatched validator excludes itself from scoring.

sandbox / evaluator
The isolated concrete-ml subprocess a validator shells into to compile, grade, and time an untrusted recipe: bubblewrap-jailed (no network, read-only filesystem, private /tmp) with an `unshare` fallback, CPU- and wall-clock-bounded, credentials stripped, returning only a small JSON verdict. The chain process never runs miner-influenced code.

score cache / results file
The validator's persistent per-`(hotkey, revision, task)` verdict store — the only channel between its score and weight processes — and the optional public per-task JSON it publishes to HuggingFace so anyone can audit every verdict, rank, and weight.

set_commitment / get_commitment
The Bittensor chain calls a miner uses to publish its `{repo, rev}` pointer and a validator uses to read it (with its commit block). Cifra's only on-chain miner→validator channel; no serving, no queries.

Yuma consensus / dTAO
Bittensor's stake-weighted aggregation of validator weight vectors, and the mechanism by which staked TAO scales a subnet's emission pool.