TrueOpen

An Open GPU Network for Verifiable and Private AI Inference

Abstract. Open-source models alone are not enough: open AI services also need an open GPU network. TrueOpen enables users to buy inference services directly from different GPU providers. The network checks inference at an economically viable cost by recomputing the output's probability characteristics with the specified model or verifying cryptographic proofs of selected computations. Payments are settled based on verification results, and GPU providers stake funds to back their obligations and face penalties for violations. Users receive outputs without waiting for verification to finish, while verification data is retained for subsequent review. Batched on-chain submissions reduce processing overhead, allowing the network to support more tasks. Content encryption and authorized key access protect inputs and outputs; administrative privileges alone do not enable decryption.

1. Introduction

Open-source models can be deployed by different providers, but the returned text alone does not establish whether the agreed model was used. Direct transactions between users and GPU providers require three things: an economical way to verify computation, rules that secure payment and hold providers accountable for violations, and protection for the privacy of inputs and outputs. Model maintainers should also be able to register models directly, without waiting for a platform to approve them.

TrueOpen allows maintainers to register models and their execution and verification requirements under public rules. GPU providers choose which models to serve and when to accept work or leave the network. Users submit tasks and place fees in escrow; Workers return inference results and submit evidence. The network verifies inference either through independent nodes checking the output's probability characteristics or through cryptographic proofs of selected computations. The first approach avoids repeating token-by-token generation, while the second reduces proving work through sampling. Each controls verification costs under its own conditions of applicability.

The blockchain settles payments based on verification results and slashes stake for confirmed violations under protocol rules. Verification data remains available throughout verification and disputes for subsequent review; stopping participation does not release a node from its existing obligations. Users receive results without waiting for verification and settlement to finish. As task volume grows, the network batches records from multiple tasks into on-chain submissions, amortizing transaction overhead to support inference at greater scale.

Inputs and inference outputs can be encrypted, with keys delivered only to authorized participants. Administrative privileges alone do not enable decryption. Developers can build AI applications using the network's inference services and manage fees, expenditures, and revenue distribution through smart contracts. Different providers can thus deliver services together without leaving model admission, computation verification, and payment processing entirely under the control of a single platform.

2. Model Registration and Task Assignment

2.1 Model Registration: Published Execution Requirements and Verification Rules

Model maintainers can register models directly under public rules, without approval from any network participant. Registration is an on-chain transaction and incurs a transaction fee. Maintainers may also set a model maintenance fee to earn revenue from subsequent services.

When registering a model, the maintainer specifies its version, execution configuration, verification rules, and pricing rules. Compute providers use these details to decide whether they can serve the model, while users use them to choose a service. Verifiers follow the registered rules to check the computation. Service prices adjust automatically to supply and demand, with the price of each task fixed when the order is placed.

A model name alone is not enough to define a task. The weights, tokenizer, input template, and numerical precision all affect the result. These details are registered as part of the model version, so each task specifies exactly which configuration to use.

Different models require different verification rules, but maintainers do not need to build an entire verification system from scratch. TrueOpen provides a common inference verification framework supporting two methods.

For Logprob Verification by Majority Agreement, model maintainers choose checks supported by the protocol. They use testing to calibrate numerical tolerances and parameters for batch-level statistical tests. These tests must distinguish normal numerical variation across GPUs from deviations caused by execution that violates the registered rules. Section 3.1.3 describes the calibration procedure.

SLP uses a shared library of verifiable operators. Maintainers build the model's computation graph from these operators and configure its verification rules. They only need to add an operator when the model requires an operation the library does not support. This avoids building proof circuits from scratch for each model. Section 3.2.5 describes how models are integrated.

Permissionless registration does not guarantee that anyone will serve a model. Demand must be matched by a credible verification scheme: maintainers must show that the scheme detects violations, accommodates normal numerical variation, and keeps verification affordable. Without convincing evidence, compute providers and verifiers may decline to support the model and users may decline to buy its service. Continued availability therefore depends on community acceptance of verification as well as market demand.

2.2 GPU Node Participation: Staking and On-Chain Performance Scores

A compute provider joins by staking funds and loading a registered model its hardware can run. It chooses when to accept or stop accepting work and can serve as a Worker generating outputs or a Verifier checking computation—but never as the verifier of its own task. Periodic on-chain heartbeats keep its availability declaration current; once the heartbeat expires, the node is excluded from new assignments.

When several nodes apply for the same task, weighted random selection combines stake with performance scores. Stake backs financial obligations; performance measures service quality. Public rules govern their calculation and relative influence, with adjustments through community governance and consistent enforcement by the protocol.

Only completed orders won under sufficient competition contribute to performance scores, limiting score inflation through self-created orders. The evidence of competition is the nodes' signed participation requests: a Builder combines their signatures into a Boneh–Lynn–Shacham (BLS) aggregate signature and submits it on-chain with the task, allowing the protocol to check whether the competition threshold for scoring has been met. Service records remain permanently on-chain. Nodes are evaluated within groups of comparable models and task sizes. The evaluation combines recent performance with long-term service history, increasing the selection chances of nodes that consistently provide reliable service.

Nodes back their service obligations with existing stake and are subject to slashing for failure to meet those obligations or confirmed cheating. Section 6.1 sets out the minimum stake requirement and its role in task eligibility.

Nodes receive tasks and invoke the inference engine through local software called Cortex. The engine performs model computation; Cortex signs receipts, retains evidence, and tracks each task's on-chain status. Cortex manages the signing keys and does not pass them to the inference engine.

2.3 GPU Task Assignment: VRF-Based Candidate Selection and Future Randomness

An order fixes the user's model, execution requirements, budget, and deadline. The protocol uses a verifiable random function (VRF), with stake and performance determining selection weights, to identify eligible candidates. Nodes check their eligibility locally; those willing to perform the task sign and submit a participation request.

The Builder collects those requests, combines their signatures into a BLS aggregate signature, and submits it on-chain with the task. Candidate selection only establishes eligibility to apply; it does not determine who will execute the task. The participant list and weights are fixed when registration closes. Randomness from a designated future block then selects the Worker from that list in a separate draw, whose result was unknown when nodes applied.

If nodes could calculate the final selection outcome when applying, they could participate only when the outcome favored them, undermining fairness. Final selection therefore uses randomness from a designated block height after the application deadline. If the list is fixed at block height h_f and the waiting interval is Δ, the randomness comes from:

$$h_W = h_f + \Delta$$

The protocol specifies this block height in advance, and its randomness is still unknown when the list is fixed. Once the randomness is published, the protocol selects the Worker using the fixed weights. Any participant can verify the result using the same list, weights, and randomness, checking for changes to the list or weights or departures from the selection rules.

Verifiability alone does not prevent manipulation. A VRF proof establishes whether randomness was generated as specified. The protocol must also prevent its producer from withholding an unfavorable result or selecting a favorable one by changing the block producer or generating another result. These safeguards, together with fixing the list before randomness becomes available, determine whether selection is fair.

Once selected, the Worker retrieves the input, checks the task requirements, and begins inference.

In Logprob Verification by Majority Agreement, Verifier selection also combines a VRF with future block randomness. The Worker first submits commitments to the output and evidence. The protocol then determines who will verify the task, preventing the Worker from knowing its verifiers in advance. Section 3.1.1 describes the selection process.

2.4 Submitting Inference Results and Evidence: Signed Receipts and Inference Speed Evaluation

The Worker delivers inference results without waiting for verification to finish, reducing the user's wait. A Builder coordinates the task, with Nexus providing its ingress and relay service. Streaming and non-streaming responses follow the same path: the former contains multiple text chunks, the latter one. Nexus applies the same signature, sequence, and content-commitment checks to both, stores the chunks, then forwards them to the user's SDK. Chapter 4 covers delivery and recovery.

At completion, the Worker hashes the full output and computation evidence in the prescribed format, includes their digests in a signed inference receipt (InferReceipt), and passes the receipt to the Builder for on-chain submission. These digests commit to the data. Verifiers can then hash the retrieved data and compare the resulting digests with the on-chain commitments to check whether the data has been altered. For Merkle-tree evidence, the recorded root also supports selective disclosure. A value and its Merkle inclusion proof demonstrate membership in the original evidence without exposing the full dataset.

The signature identifies the submitter, and the on-chain record fixes its commitment. Even if the Worker alters its local data, the altered content will not match the original commitment. A commitment binds the data, however; it does not establish computational correctness. Further

verification is still required.

When encryption is enabled, the Worker encrypts the inference output before delivery. The Builder checks signatures, chunk ordering, and ciphertext commitments, then stores and relays the ciphertext. After decryption, the user's SDK checks the output's content commitment. Storage and relay nodes can therefore detect changes to the received ciphertext without reading the plaintext. Chapter 9 describes key generation and distribution.

DA retains the complete output and computation evidence, with access governed by task permissions and the verification phase. In Logprob Verification by Majority Agreement, verifiers can retrieve the input and output needed for computation. They cannot read the Worker's logprobs used for comparison until they have submitted commitments to their own computed values. Uploading evidence for storage and revealing its numerical values are separate steps. Section 3.1.2 describes the reveal rules.

On-chain records provide a verifiable basis for evaluating inference speed. Timing starts when the chain confirms the Worker’s selection and ends when it accepts the inference receipt. Within each task bucket, this elapsed time is divided by the number of output tokens to obtain the average time per token used in calculating the speed score. It includes computation, data transfer, and on-chain delays, rather than GPU compute time alone. The receipt is submitted separately, without waiting for the full evidence upload.

After receiving the data, the Builder checks it against the commitments in the receipt. If they match, it issues a signed storage acknowledgment and assumes responsibility for retaining the data. Data needed for subsequent verification must remain retrievable. The storage signature records who is responsible; retrieval checks, audits, and challenges test continued availability.

3. Inference Verification: Logprob Verification by Majority Agreement and Cryptographic Verification with Sampled Layerwise Proofs (SLP)

A signature identifies the party that submitted data, and an on-chain commitment makes subsequent changes detectable. Neither establishes that the model computation was correct. Inference verification checks whether the node used the agreed model and execution configuration. Factual errors or hallucinations produced by the specified model are not, by themselves, evidence of node misconduct.

TrueOpen supports two approaches. In Logprob Verification by Majority Agreement, independent nodes use the specified model to compute values for comparison. SLP generates cryptographic proofs for selected computations in quantized models; floating-point models instead undergo commitment-based numerical audits. Both approaches must bind the evidence to the same task and detect violations at an acceptable verification cost. Model registration fixes the checks, numerical tolerances, and batch acceptance criteria. Verifiers apply those rules, and the protocol uses their results to determine the outcome and settle payments.

3.1 Logprob Verification by Majority Agreement

During generation, the model repeatedly computes a next-token distribution from the input and the tokens generated so far. The decoding rule determines which token is chosen next. After generation, all those prefixes are already known. A verifier can therefore compute the next-token distributions for those prefixes in parallel with one prefill pass through the specified model. This yields the probabilities and ranks of the returned tokens without generating them again sequentially.

This provides an independent basis for comparison. The token log probabilities reported by the Worker must agree with those computed using the specified model within the registered tolerances, and the returned tokens must satisfy the registered candidate-selection rules. Numerical tolerances and batch-level statistics account for normal numerical variation across hardware. We first explain how verifiers compute independently, then how the computed values are used to reach a decision.

3.1.1 Random Selection of Verifiers

A Worker that could choose its own verifiers might collude with them to bypass the computation rules. The protocol therefore requires the Worker to commit to its output and evidence before the final verifiers are selected. It cannot then replace those materials in response to the selection outcome.

Candidate verifiers are selected through a verifiable random function (VRF) from nodes that meet the stake and performance requirements and whose online declarations are still valid. They must also support the task's model and verification configuration and remain eligible under the penalty rules. Willing candidates submit signed participation requests. Once registration closes and the participant list and weights are fixed, randomness from a designated future block selects three verifiers by weighted sampling without replacement. The task's Worker is excluded.

These are two separate draws: the first establishes eligibility to apply; the second chooses verifiers from those who actually applied. Selection uses the stake and performance rules in Section 2.2 rather than a separate scoring system. Anyone can reproduce the selection from the fixed list, weights, and randomness. Resistance to collusion depends on the selection weight controlled by malicious nodes and on the randomness source's resistance to manipulation—not simply on how many nodes exist.

3.1.2 Independent Prefill Computation and Commit–Reveal

A single pass over the input and complete output, using the user's specified model, weight version, and execution configuration, yields the logprobs, ranks, and top-k candidate distributions for verification. Consider a Worker output of A, B, C. To assess B, the model predicts the next token from the original input and A; the verifier looks up B's probability and rank in that distribution. Neither B nor C influences that prediction. The causal mask restricts attention at each position, allowing parallel prefill computation to preserve generation's causal dependencies.

Verification uses the original token sequence recorded by the Worker during generation. Different token sequences can produce the same text while having different probabilities and ranks, so re-tokenizing the displayed text is not a valid substitute. The verifier must also check that decoding the recorded sequence produces the text delivered to the user.

To prevent verifiers from copying the Worker's comparison values, those values remain inaccessible until verifiers have committed to their own results. Each verifier encodes its metrics and evidence as specified by the protocol and hashes them. It then combines that digest with the task information and a secret random salt to produce an on-chain commitment. The salt prevents others from guessing the committed values by enumerating possible answers before the reveal. During the reveal phase, the chain recomputes the commitment and rejects any mismatch.

```
commit_hash = SHA256(
  frame("SINGA_RESULT_COMMITMENT_V2")
  || frame(chain_id) || frame(task_id) || frame(task_hash)
  || frame(u32be(verify_round))
  || frame(verifier_operator_address_bytes)
  || frame(result_payload_hash) || frame(salt)
)
```

`chain_id`, `task_id`, and `task_hash` bind the chain, task identifier, and task contents; `verify_round` and the address bytes bind the verification round and verifier identity. `result_payload_hash` is the digest of the protocol-encoded metrics and evidence; it also binds the inference receipt, model configuration, and generation parameters. Each commitment uses a fresh 32-byte `salt`, kept secret until the reveal. `frame` prefixes each field with its length as an 8-byte big-endian integer, avoiding ambiguous concatenation. `u32be` encodes a 4-byte big-endian integer, and `||` denotes byte concatenation. See the Result Commitment Implementation Notes for the complete fields and encoding.

Selection randomness and commitment salts have different roles. The protocol uses a future block's VRF beacon to select nodes from the fixed candidate list; once published, it lets participants verify the draw. A salt is generated independently on the verifier's own machine and

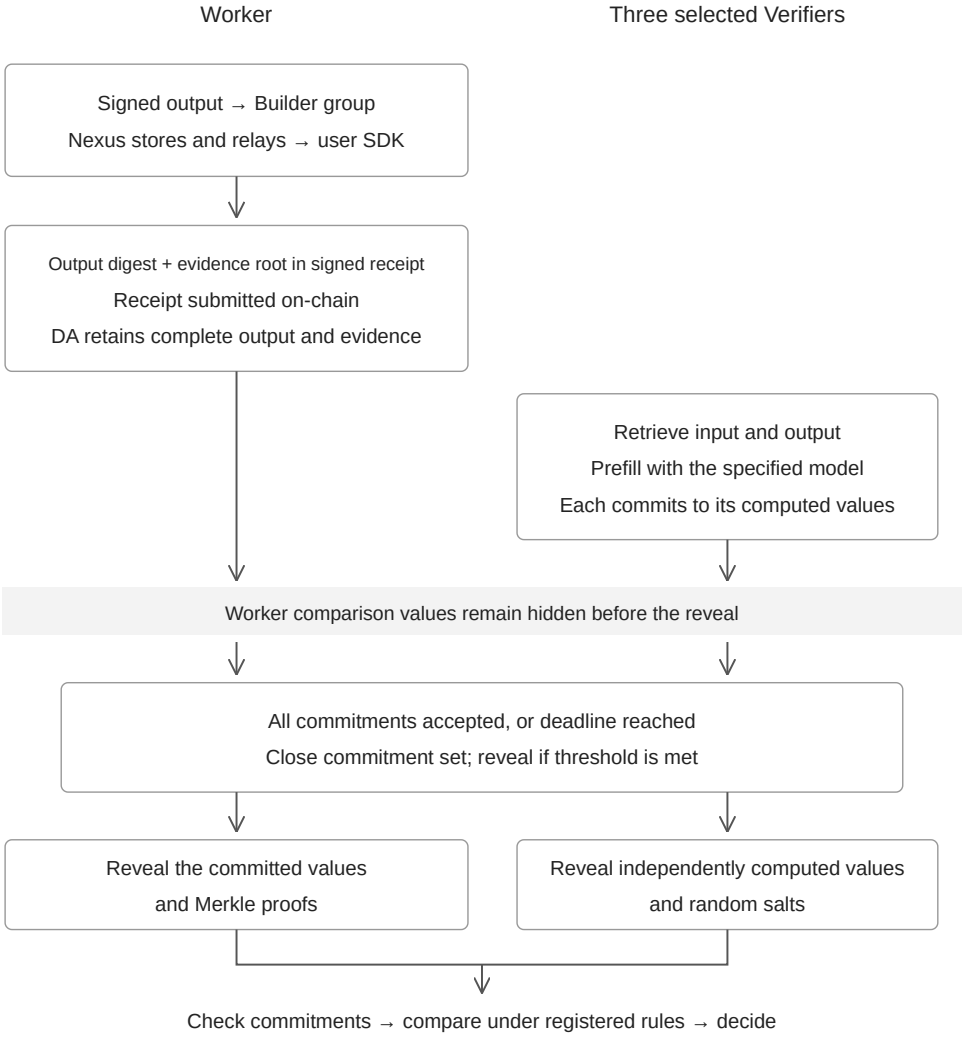
remains secret until the reveal. It must not be derived from the beacon or other public information. Selection randomness is not an input to the commitment formula above.

Verification covers every generated token, rather than a random sample within the output. Each position must have a corresponding metric record, and missing records must be accounted for. A verifier cannot change the scope by omitting tokens that are unlikely to pass. The batch statistics in Section 3.1.3 combine multiple tasks; they are distinct from token sampling within a task.

For encrypted tasks, the token sequence needed for independent computation and the Worker's comparison values are encrypted separately. Verifiers first receive the input, output, and original token sequence, compute their own values, and commit to them. Only in the reveal phase does the Worker release the separate key for its comparison values. Storing ciphertext in advance does not make those values available early. Chapter 9 describes the key distribution scheme.

The round's commitment set closes when all selected verifiers' commitments have been accepted on-chain or the commitment deadline expires. Two accepted commitments do not trigger an early reveal to a third verifier that has not yet committed. If fewer than two valid commitments exist at the deadline, verification fails. Otherwise, the round enters the reveal phase, after which no new commitments are accepted for that round.

The Worker then supplies the values and proofs corresponding to its original evidence commitment; verifiers supply their own values and salts. Each submission is checked against its commitment before numerical comparison begins. Verifiers must retain the original data and salts so that they can complete the reveal. By the time either side sees the other's values, neither can change its own answer.



Output delivery precedes the reveal of verification values. Both sides are already bound by their commitments when they see each other's values.

3.1.3 Detecting Noncompliant Computation: Numerical Comparisons and Batch Statistics

After the reveal, verification must distinguish ordinary numerical variation from a violation. Normal differences in logprobs occur even when the same model runs on different GPUs of the same type; requiring exact equality would reject honest computation. Maintainers therefore calibrate the rules against both compliant and noncompliant executions across hardware and inference engines, seeking to separate normal variation from violations.

The first comparison measures how far the Worker's logprobs deviate from those computed with the specified model. Let ℓ^{W}_t and ℓ^{V}_t denote the two logprobs at output position t. Over N valid positions, their mean absolute difference is:

$$D_{mean} = (1/N) \Sigma_t \left|\ell^{W}_t - \ell^{V}_t\right|$$

N is the number of valid comparison positions. ℓ^{W}_t and ℓ^{V}_t are the Worker's and Verifier's logprobs for output token t. A smaller D_mean indicates closer agreement, but acceptance still depends on the registered threshold and the other checks.

A mean can conceal a few large errors, so the rules also examine the 95th and 99th percentiles of the differences. The token rank-change rate tests whether relative preferences have shifted. Where enabled by the model configuration, top-k set overlap and differences between probability distributions further test agreement among high-probability candidates. All these metrics come from the same prefill pass; their definitions and individual roles are given in the supporting technical documentation.

Matching probability data does not establish that the returned tokens satisfy the agreement. Where the registered rules require candidate checks, verifiers also check each returned token against the specified model's allowed candidates. For input x, generated prefix $y_1 \ldots y_{t-1}$, and registered candidate limit k_gen, the condition is:

$$\text{rank\_M}(y_t \mid x, y_1, \ldots, y_{t-1}) \leq \text{k\_gen}$$

M is the model specified by the user, x is the input, y_t is the token under examination, and $y_1 \ldots y_{t-1}$ is its actual generated prefix. The registered parameter k_gen limits the number of allowed candidates. The token must rank among the model's k_gen highest-probability candidates when conditioned on that prefix.

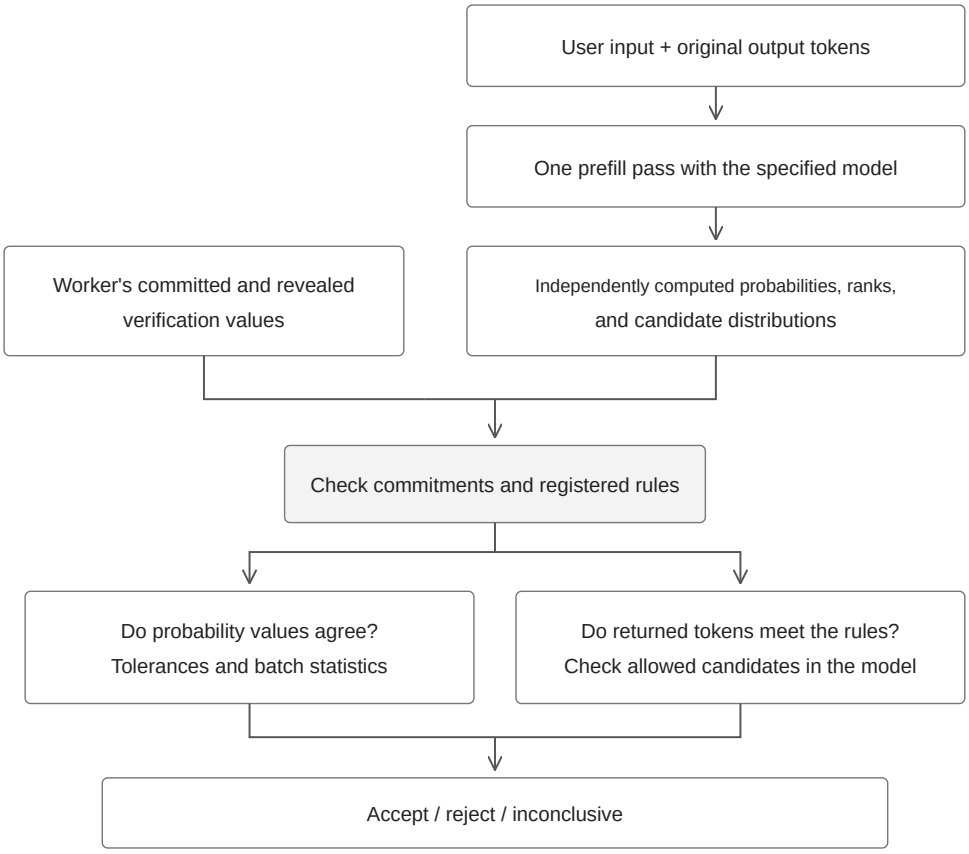
This check ties the decision to what the user actually received. A token clearly outside the allowed candidate set is grounds for rejection. Positions near a numerical boundary are handled under the registered tolerance; they do not pass merely because other positions are normal.

Matching probabilities do not by themselves prove the specified model produced the output. A cheating Worker can generate tokens with a cheaper model and then run a single prefill pass through the specified model to obtain those tokens' genuine logprobs, submitting them as evidence; because the values then agree with the specified model, the mean-absolute-difference check cannot detect the substitution. Detection instead rests on the candidate check: some tokens produced by a smaller model fall outside the specified model's top-k. To make this check a strong constraint, the network enforces a bounded sampling top-k as a registered execution requirement, setting k_gen to that bound—every token of a compliant output must lie within the specified model's top-k, while the deviating tokens of a substitute model are reliably caught by the per-token candidate check and batch statistics.

Individual tasks can still be affected by incidental variation. The protocol therefore groups tasks from the same Worker with the same model and verification configuration into statistical batches, fixing batch membership before verification. Model tests determine batch size, the minimum number of valid samples, and acceptance and rejection criteria. Batch statistics assess overall deviation, while per-token checks continue to constrain violations at particular positions [15].

A result passes only if all acceptance criteria are met. It is rejected if a registered rejection condition is met. Insufficient evidence or an intermediate outcome remains inconclusive and triggers further checks under the rules. Missing data, computation failures, and unavailable evidence are handled separately: discarding failed samples cannot turn a result into a pass. Metrics, tolerances, and batching rules are fixed before the task starts, preventing verifiers from choosing a favorable test after seeing the result.

Once the statistical checks are complete, the protocol examines the three selected verifiers' valid submissions. At least two must agree in accordance with the rules to establish majority agreement and the corresponding billable workload. Batch statistics address numerical variation; independent verifiers constrain reliance on any single node's judgment. Neither substitutes for the other.



Each verifier performs these checks independently. At least two of the three selected verifiers must submit valid, agreeing results for an order to reach majority agreement. MoE models also require expert-routing checks. Values are revealed only after the commitment set closes, as described in Section 3.1.2.

3.1.4 Joint Verification of Expert Routing and Outputs in MoE LLMs

The preceding probability and candidate checks examine the generated output. Dense models use a fixed dense computation structure; mixture-of-experts (MoE) models route each token to a subset of experts, potentially choosing different experts for different tokens. Verification must therefore also check whether the selected experts follow the specified model's routing rules.

During generation, the Worker records expert identifiers and their slot order and commits to that evidence. The Verifier obtains an independent routing trace from a prefill pass with the specified model. After the reveal, the traces are aligned by global token position and compared over their common coverage, preserving slot order. Different trace boundaries are thus not mistaken for different expert choices.

Expert selection is discrete, so normal numerical variation can change which expert is chosen for an individual routing slot. The scheme accommodates this by measuring mismatches across a batch: the total number of mismatching slots is divided by the total number actually compared. If sample i contributes m_i mismatches across e_i compared slots, the routing mismatch rate is:

$$R = (\Sigma_i\ m_i) \ / \ (\Sigma_i\ e_i)$$

m_i counts slots where the Worker and Verifier chose different experts; e_i counts all slots compared in sample i. R is the batch-wide mismatch rate. For example, 50 mismatches among 1,000 compared slots give R = 5%. Acceptance depends on the routing threshold calibrated during model registration.

Weighting by the number of compared slots avoids giving short and long samples equal influence. Model tests determine which layers to compare, the required sample count, and the allowed routing mismatch. For example, the described Qwen3.6-35B-A3B configuration checks first-layer routing to reduce the extent to which numerical differences accumulated in later layers interfere with distinguishing normal variation from violations. Configuration details and experiments remain in the supporting documentation.

A batch is accepted only if it satisfies both the routing threshold and the output candidate rules in the preceding section, with sufficient evidence and no unresolved positions. One checks which experts were used; the other checks whether the resulting tokens meet the requirements. Passing either alone is insufficient.

3.1.5 Verification Cost and Scope

Verification checks the generated sequence in parallel with a single prefill pass, rather than generating it again token by token. In earlier tests of Qwen2.5-7B, the combined GPU time for three verifiers was **1.9%–14.8%** of the time needed to generate the output once under the tested configurations. In single-request tests of Qwen3-8B on an RTX 4090 and Qwen3-32B on an H100 NVL, a single verifier took approximately **0.85%–8.42%** of the inference generation time for outputs of 1,024–8,192 tokens. The two setups showed similar timing ratios, supporting the cost advantage of parallel prefill verification across dense models with different parameter counts. This cost advantage comes from the difference between autoregressive, token-by-token generation and parallel verification of a known sequence; it is not specific to one model size. The same principle applies to dense models with other parameter counts when model architectures, precision settings, inference frameworks, batching strategies, and hardware are comparable. Multi-GPU parallelism, quantization, and inference optimizations affect the exact ratio, but for long outputs generated autoregressively at small batch sizes, verification through parallel prefill typically still uses substantially fewer compute resources.

These checks assess whether output probabilities, candidate choices, and applicable routing characteristics satisfy the registered rules. They are not a proof of every step of generation. Detection depends on the selected checks and calibration data; majority agreement cannot extend their scope.

3.1.6 Review and Correction After Majority Agreement

Majority agreement does not make a decision irreversible. The chain records commitments, signatures, and decisions from the Worker and verifiers, while DA retains the corresponding data for the required period. Participants with the necessary read permissions can run prefill with the specified model to check the values and original decision. If they find an error that meets the challenge criteria, they can submit evidence and request a settlement correction and penalties for the responsible parties. Agreement among a majority therefore does not prevent a wrong decision from being discovered and corrected.

3.2 Sampled Layerwise Proofs (SLP)

Inference consists of operations such as matrix multiplication, normalization, activation functions, and routing. Each defines a relationship between its inputs and outputs. A proof system expresses those relationships as mathematical constraints. The prover uses the computation trace to generate a proof; the verifier checks it without repeating all the proven computation [8]. Assuming a secure proof system and a correct verification implementation, the probability of accepting a proof for a computation that violates the constraints is negligible [16].

A proof must also refer to the inference the user purchased. TrueOpen binds model weights, inputs, layer computations, and the delivered output through commitments, so that a proven operation uses the specified model and connects to the preceding and following computations of the same task. The guarantees here concern computational correctness, not zero knowledge. Not needing every intermediate value for verification does not imply that the proof hides those values, and the node performing inference still needs access to the input.

For quantized models that follow the fixed-point specification, a designated verification program checks the proof. Anyone with the proof, required public information, and corresponding verification program can verify it independently; the blockchain still applies protocol rules to accept results and settle payments. Floating-point verification provides no such computation proofs. It requires compute nodes to perform numerical audits and evaluate the results under the registered rules, as described in Section 3.2.2.

Proof verification can be much cheaper than repeating the computation, but proving an entire inference still consumes substantial compute [9][10]. TrueOpen organizes the computation by layer, proving fewer intermediate operations while retaining mandatory input and output checks. The following sections explain the commitments, selection of proof locations, model-specific checks, and security conditions.

3.2.1 Commit to the Computation Before Selecting Proof Locations

The computation is partitioned into chunks along layer boundaries. Intermediate values, called activations, connect these chunks. The Worker creates polynomial commitments to each chunk's inputs and outputs, with adjacent chunks sharing boundary commitments and a hash chain fixing their order. Model-weight commitments are prepared in advance and reused across tasks for the same version. Each task fixes a single valid computation commitment before future randomness determines the check locations. Replacing the commitment or retrying the task does not permit a new draw. Computation required to bind the input and output is always checked; other chunks are sampled within the specified scope.

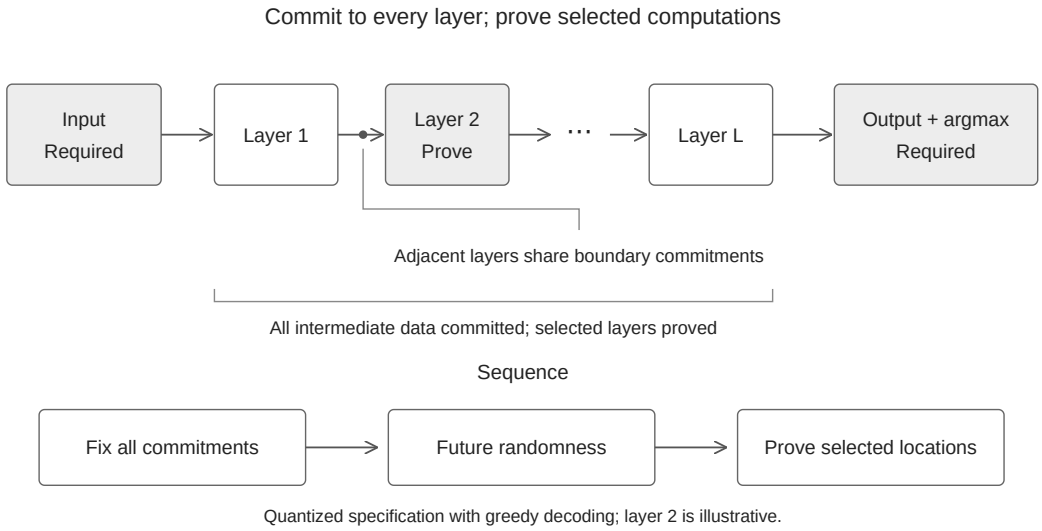
For a selected layer ℓ, the proof checks:

$$a_\ell = F_\ell(W_\ell, a_{\ell-1})$$

Here W_ℓ denotes that layer's weights in the specified model, $a_{\ell-1}$ and a_ℓ its committed input and output activations, and F_ℓ all operations in the layer. The proof establishes both correct computation and consistency with the registered weights and original commitments. Shared boundary commitments prevent independently correct fragments from unrelated executions from being spliced together [10].

The endpoints must also bind the trace to the user's input and delivered output. Token embeddings and the output projection are therefore always checked. The embedding binds the input; the projection connects the final activations to logits, whose argmax determines the returned token. This directly binds the token for greedy decoding. With temperature-based or stochastic sampling, proving the logits alone is insufficient: the sampling randomness must also be committed, and the proof must establish that the token follows from those logits, decoding parameters, and randomness. Unless this relation is proved, the proof does not establish that the actual returned token follows from the computation.

MoE models additionally require checks of expert scores, top-k routing, and the correspondence between token dispatch and the combination of expert outputs. The model commitment covers all experts; proofs of expert computation cover those actually activated. Randomized verification proves only the specified scope. Its ability to detect errors across the full computation depends on the coverage conditions in Section 3.2.4.



Commitments bind the data; proofs check the computation. The data for an unselected layer is committed, but that commitment is not a complete proof of the layer's computation.

3.2.2 Verification Across Numerical Representations

For fixed-point quantized computation, number representation, scaling, rounding, and arithmetic ranges can be specified precisely and checked by the proof system. The claim is that the quantized model followed that specification, not merely that its output was close to another GPU's. Integer weights alone are insufficient: the actual arithmetic must follow the registered fixed-point rules.

Matrix multiplication accounts for much of model computation. As a cheaper supplementary check, the scheme first commits to X, W, and the claimed product Y, then draws a random vector r and tests:

$$X(Wr) = Yr$$

Matrix–vector products cost less than recomputing the full matrix product. If Y = XW, the identity holds; otherwise, a random vector may expose the error. Under the standard exact-arithmetic assumptions, t repetitions with independent, uniformly sampled binary vectors limit the probability that an incorrect product passes every check to at most 2^{-t} [18].

Commitments alone cannot perform this test: the checking node needs the actual X, W, and Y. It must hold the layer's weights, retrieve its input and output activations from DA, and check them against the fixed commitments. For a quantized model, Y is the integer accumulation result before requantization. Subsequent rules separately check requantization's scaling, rounding, and truncation. This differs from proof verification, which needs only the public verification context: matrix checks require weights, activations, and permission to read them. They can supplement proofs by checking layers not selected for proof generation [11]. The combined benefit depends on coverage and correlation, as discussed in Section 3.2.4.

Floating-point execution does not support the same exact-equality requirement. Normal variation from hardware, evaluation order, and batching is measured through a residual:

$$\delta = X(Wr) - Yr$$

Model tests calibrate the residual metric, tolerances, and batch decision rules [12].

Layerwise proofs in this scheme cover integer fixed-point arithmetic; no layer in the floating-point path receives a computation proof. Instead, that path uses commitments to weights, inputs, inter-layer activations, and outputs, together with matrix checks and tolerance-based decisions on the committed data. Commitments prevent later substitution; tolerances determine acceptance. The

exact-arithmetic bound of 2^{-t} does not carry over to tolerance-based tests, whose calibration must assess the noncompliant deviations they might accept. Layerwise checks assess whether each layer's computation meets the registered tolerances. Bounding the numerical error in the final output also requires an analysis of how errors propagate between layers [13].

Verification must also check that the returned tokens follow the agreed decoding rules. Floating-point models use calibrated numerical audits to detect noncompliant computation. Tasks that also require a bound on the error in the final output need additional stability and error-propagation analysis.

3.2.3 Costs of Sampling and Batched Proofs

Users first escrow their fees, and the network selects a Worker under the preceding rules. The Worker runs the model, returns the output, and retains intermediate computation data. Orders completed by that same Worker within a specified time window form a batch under the registered verification rules. Closing the batch fixes its member orders, program version, and data commitments. A designated future randomness beacon then selects check locations, and the prover generates proofs for the selected segments. The protocol checks that the proofs are valid, refer to orders in the batch, and cover every mandatory location. Disputes and settlement then follow the protocol rules.

Valid evidence of a computation violation in one order causes the whole batch to be treated as fraudulent. Missing proofs and timeouts are handled under their respective failure rules. Batch-wide liability increases the loss upon detection, but does not change the detection probability of the random checks themselves.

Sampling reduces the number of chunks that require proofs, while packing multiple requests amortizes proof-generation overhead. Weight preprocessing can also be reused across tasks. Each task still incurs the costs of data commitments, mandatory endpoint proofs, and evidence retention. Total proving cost therefore cannot be estimated simply from the fraction of chunks selected. Verifying a proof for a model that follows the quantized specification requires neither model weights nor a GPU; generating it still requires the proving computation. Matrix checks and floating-point audits, by contrast, require checking nodes to hold weights, obtain activations, and perform calculations.

In the tested TinyLlama configuration, SLP compared full and sampled proving on the same execution trace. With the mandatory input and output chunks retained and five intermediate chunks randomly selected, sealing and proving time fell from 1,256.1 to 276.8 seconds, a reduction of approximately 78% [2].

In the tested TinyLlama configuration, generating a packed proof for three requests took 116.4 seconds, compared with a total of 295.6 seconds for separate proofs—a reduction of approximately 61% [2]. This result shows that packing multiple requests can amortize proof-generation overhead.

The packing experiment already used sampling, showing that the two techniques can be combined. However, the experiments used different sampling configurations, so their speedup factors cannot simply be multiplied. The combined reduction must be measured under a consistent configuration.

The cost benefits of sampling and packing are not unique to TinyLlama. Models with different parameter counts are expected to achieve similar relative cost reductions when their architectures, proof frameworks, sequence lengths, sampling rates, and batch sizes are comparable. The actual reductions still need to be confirmed through measurement.

3.2.4 Detection Probability and Economic Deterrence

For models that follow the fixed-point specification, substituted weights violate the model commitment, substituted inputs violate input binding, and substituted output tokens under greedy decoding violate the decoding constraint [14]. Incorrect routing violates expert-selection and dispatch constraints. These checks are mandatory. Errors in intermediate computation must be covered by the random checks, so their detection probability must be stated separately.

Suppose b of M eligible sampling units contain errors detectable by the corresponding proofs. Uniformly sampling s units without replacement hits at least one erroneous unit with probability:

```
P = 1 − C(M−b, s) / C(M, s)
```

M is the number of eligible locations, b the number containing detectable errors, and s the number sampled without replacement. C denotes a binomial coefficient. Assuming uniform sampling, subtracting the probability of missing every error from one gives the probability of hitting at least one. This is a coverage probability; the proof system's own soundness error must be accounted for separately.

If skipping computation corrupts multiple units, the chance of sampling an erroneous unit increases. With only one erroneous unit, that probability is s/M. Matrix checks provide an additional detection path, whose combined effect depends on coverage and correlation.

To reduce verification overhead, the network can check every order or first sample orders and then randomly select computation chunks within them [15]. Broader coverage increases both the chance of detecting errors and the verification workload. The protocol adjusts the order sampling rate, the number of sampled chunks, and penalty parameters together, taking verification costs, potential gains from cheating, and enforceable penalties into account so that cheating is unprofitable.

When a task requires a more complete computation guarantee, the quantized path can generate proofs for every layer from the already committed data. The floating-point path remains subject to its tolerances and statistical decision rules.

3.2.5 Reusing Verifiable Operators Across Models

Matrix multiplication, attention, normalization, and activation functions recur across models. TrueOpen implements these as a shared library of verifiable operators checked by a general-purpose proof system, so maintainers do not need to build proof circuits from scratch for each model.

Existing operators can be composed according to a model’s computation graph. An unsupported operation requires a computation definition, proof relation, numerical rules, and tests. Once added to the shared library, the operator can be reused by other models. Reuse still requires checking that the proof constraints accurately encode the model’s quantization, rounding, normalization, and other arithmetic rules.

4. Prompt Output Delivery, Followed by Verification and Settlement

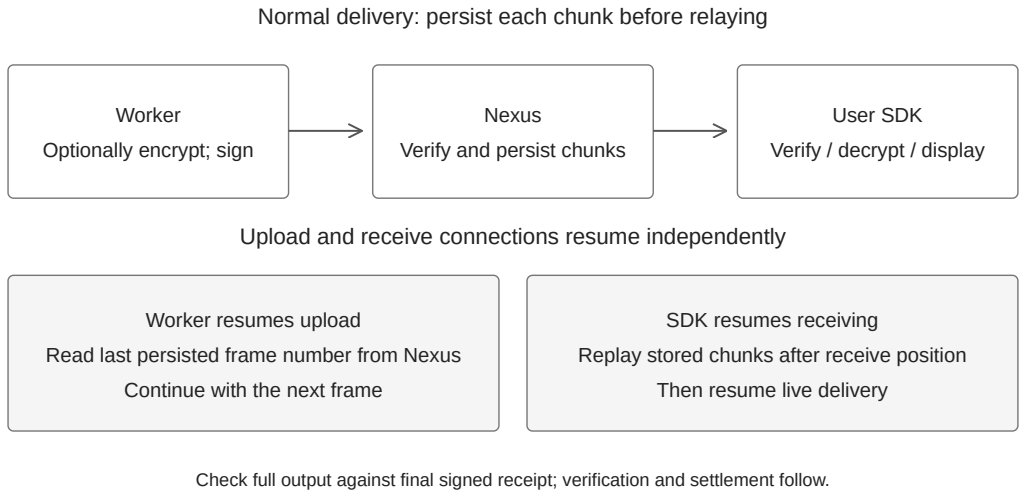
Waiting for a batch to fill, verification to finish, and settlement to complete would add unnecessary delay before users receive their results. TrueOpen separates delivery from those later steps: it begins returning results as soon as the inference engine makes them available. Node earnings remain subject to verification and settlement rules.

Both streaming and non-streaming responses use the same delivery path. A streaming response contains multiple text chunks; a non-streaming response contains one. Both use the same signatures, ordering checks, and content commitments. The Worker sends numbered, signed chunks with cumulative commitments to the task's Builder group. Nexus first checks that the sender is the assigned Worker, then checks each signature, sequence number, and cumulative commitment. Only after durable storage does it forward the chunk to the user's SDK. Persisting before forwarding leaves a delivery record from which interrupted connections can resume.

Encrypted inference results can still be streamed. The Worker encrypts and signs chunks as they are generated. Nexus checks the integrity and ordering of the ciphertext, while the user’s SDK verifies, decrypts, and checks the content before displaying it. After a disconnection, retransmission uses the original chunks and sequence numbers to prevent duplication, omissions, or mixing with another output stream. Once transmission is complete, the length and commitment of the full result are checked.

If a chunk fails the signature, ordering, or cumulative commitment check, Nexus stops accepting the stream and reports the reason to the Worker. Previously stored valid chunks are retained for subsequent recovery and review.

After a connection is interrupted, transfer resumes from the last confirmed position. The Worker continues uploading from the point Nexus has already stored. When the user's SDK reconnects, Nexus first sends the chunks it has missed, then resumes live delivery. This avoids retransmitting the entire result while ensuring that content produced during the interruption is not omitted.



Uploads resume from Nexus's durable-storage position; downloads resume from the SDK's receive position. Users need not wait for verification or settlement. With encryption enabled, Nexus checks ciphertext and the SDK checks plaintext after decryption. Reconnection does not remove the online obligations in Chapter 9.

Early chunks must remain bound to the complete output later verified. Cumulative commitments advance with each chunk, and every frame signature binds the content up to that point. The final signed receipt binds the entire stream. Nexus compares the receipt against the stored object's commitment—the ciphertext commitment in encrypted mode—and issues a storage acknowledgment if they match. Otherwise, it retains both chunks and the conflicting records for investigation. The full output is also retrievable through the data interface. Frame signatures and final commitments establish which output was delivered. The verification methods in Chapter 3 check whether it was produced by the agreed model computation.

Neither streaming nor non-streaming delivery requires users to wait for subsequent verification and settlement. Chapter 9 describes authorization and delivery for encrypted outputs.

Delivering an output does not immediately release earnings. The protocol retains the evidence, relevant earnings, and collateral throughout verification, dispute resolution, and settlement. Confirmed cheating triggers refunds and penalties. Delivery, verification, and settlement are recorded separately, allowing applications to distinguish content already received from a subsequently confirmed verification result.

5. Data Availability

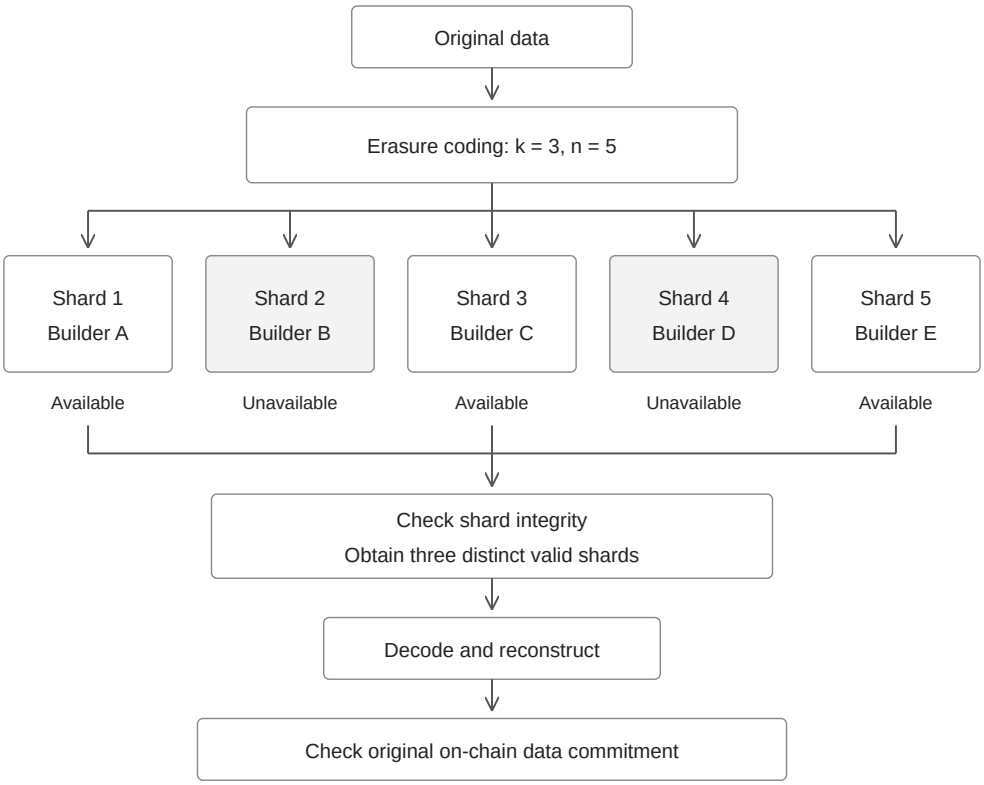
Verification and disputes require the original data. An on-chain commitment can detect substitution, but cannot make missing data available. Evidence held only by the Worker might disappear when it leaves. TrueOpen therefore has Builder groups retain inputs, outputs, and computation evidence. The purpose of its data availability (DA) mechanism is to let verification continue even after the Worker stops participating.

Participation and selection are documented jointly by on-chain records and evidence retained locally by Builders. Reviews compare signed participation requests, receipt records, and on-chain selection records to check the list and selection procedure. This evidence is managed separately from the inputs, outputs, and computation evidence held in DA.

Builders register their identities and service endpoints, post the required admission bond (BuilderBond), and satisfy candidate eligibility and penalty-status requirements. Eligible candidates enter the Builder set based on on-chain verifiable records of service, participation, and violations. The bond secures their obligations; adding to it does not buy a higher ranking. Periodic set updates balance service continuity with admission of new members. DA duties are shared by this set, without a separate storage-node eligibility class.

Builders must continue to meet the protocol's minimum stake requirement. Those below the threshold receive no new tasks, as described in Section 6.1. Storage capacity also limits the amount of data assigned to them.

Replicating every object in full at every node would multiply storage and transfer costs. Instead, a producer uploads once to an ingress Builder, which erasure-codes the data into data shards and parity shards and distributes them to the responsible Builders. Each stores its assigned shards rather than a complete replica. With n = k + m shards—k data shards and m parity shards—any k distinct valid shards can reconstruct the original under the coding assumptions. Ideal storage overhead is approximately n/k times the original size. The storage configuration determines the actual node and shard counts.



This example uses a three-of-five erasure code with one shard per Builder. If shards 2 and 4 are temporarily unavailable, integrity-checked shards 1, 3, and 5 suffice to reconstruct the original. The five shards are not five full replicas. The protocol determines actual shard counts, placement, and recovery thresholds.

Possession of shards alone does not prove that they reconstruct the correct data. The protocol separately commits to the source data and encoded output. An independent Builder checks their correspondence, and storage nodes sign acknowledgments identifying the shards received and retention deadline. Together, these checks and signatures form a storage certificate. A reader reconstructs the data from sufficient shards and checks the original content commitment.

Storage must tolerate offline nodes. If q distinct valid shards have confirmed storage and the system must tolerate f of them becoming unavailable, recovery requires:

$$q - f \geq k$$

q counts distinct valid shards confirmed as stored; f is the number allowed to become unavailable; k is the minimum needed for reconstruction. After subtracting unavailable shards, at least k must remain. The count is of shards, not signatures: a node holding several shards can take all of them offline at once.

A one-time signature cannot guarantee continued storage. Random audits require nodes to return specified pieces and proofs. As available shards decline, repair starts before the recovery threshold is reached. The original node's storage obligation is released only after a replacement checks and accepts the reconstructed shards. Repair and Builder-set rotation leave the original commitment unchanged, preserving the evidence for later verification.

If data cannot be read, a challenge requires the storage node to return specified pieces and proofs by a deadline. The protocol determines responsibility from those proofs and on-chain submissions. Repair need not wait for penalties to conclude. If recovery ultimately fails, refunds and storage liability are handled under the rules.

Task budgets fund storage, retrieval, and repair, with payments released as service is delivered and relevant earnings and collateral retained for accountability. Retention covers verification and accepted disputes; deletion is allowed only when these obligations end. Commitments, liability records, and settlement records remain on-chain. With optional content encryption enabled, Builders encode, store, and repair encrypted inputs, outputs, and evidence without reading their contents. Chapter 9 distinguishes read authorization from decryption permissions.

6. Settlement and Economic Incentives

Users need confidence that payment buys the agreed service; nodes need confidence that completed work will be paid for. TrueOpen secures both payment funds and the collateral needed to enforce obligations before work begins. Settlement then follows the service and verification results, rather than either party's willingness to pay afterward.

6.1 Fee Escrow and Minimum Stake Requirements

The order fixes service terms and a price ceiling. Once accepted on-chain, its budget is placed in escrow. Providers no longer depend on users agreeing to pay after receiving the output, and users are protected from later price increases. Settlement reflects confirmed service and workload, with unused budget returned to the user.

Nodes stake funds to back their service obligations. Only nodes that meet the protocol’s minimum stake requirement are eligible for task assignment; those below the threshold receive no new tasks. Failure to meet service obligations or confirmed cheating triggers slashing under the protocol rules. Minimum stake requirements and penalty parameters are adjusted according to public rules.

Optional encryption adds key-distribution and data-processing costs, disclosed before task submission. If an encrypted task stops because the user fails to provide required authorization, node payments follow Section 9.4. Such an authorization failure is not automatically classified as a node computation failure.

6.2 Payment for Completed Service

With funding secured, participants are paid for the work they actually complete. Workers earn revenue for inference that passes verification. Verifiers earn revenue for valid checks, including correctly detecting and rejecting cheating; otherwise they would be incentivized to approve invalid results. Missing, invalid, or disproven verification earns no corresponding verification fee.

Model maintainers earn maintenance fees under their published registration terms. Builders earn fees for coordination and storage and receive reimbursement for necessary on-chain submission costs under the rules. Consensus Validators receive protocol-allocated transaction fees

(gas fees). Network maintenance fees go to the public treasury and are accounted for separately from model-maintainer revenue. Each participant decides whether to continue serving based on whether revenue covers its costs.

6.3 Deferred Settlement and Deterrence of Cheating

An output may have been delivered while verification or disputes remain open. Releasing all earnings and collateral at that point could make later losses impossible to recover. The protocol therefore retains the relevant funds until the obligations end. Storage earnings likewise become available as the retention obligation is fulfilled.

Once batch fraud is confirmed, the protocol refunds users in the batch from escrow, cancels the responsible nodes' corresponding service earnings, and slashes their stake according to the protocol rules. Refunds and penalties are separate: the former return users' funds, while the latter are borne by the responsible nodes. Nodes whose fraud is established by the evidence are suspended from accepting tasks, with findings and penalties recorded on-chain.

Penalties are intended to outweigh the gains from skipping computation or fabricating results. Let G be the additional payoff from undetected cheating relative to honest execution, P the detection probability, and D the net loss upon detection relative to honest execution. Cheating has negative expected incremental payoff when:

$$(1 - P)G - PD < 0$$

With probability 1−P, cheating escapes detection and yields an extra gain G. With probability P, it incurs a net loss D. The expression compares cheating with honest execution; D is a net loss, not a penalty multiplier.

The order sampling rate and the verification coverage within each order jointly determine the chance of detection. Stake makes penalties enforceable. Batch-wide penalties increase the loss after detection, not the probability of sampling an erroneous location. Both must be set together so that honest service is more profitable than cheating.

6.4 Correcting Verification Errors

Initial decisions can be wrong and must remain open to evidence-based review. Overturning a false acceptance corrects settlement and assigns responsibility to the Worker and relevant verifiers according to the evidence. Overturning a false rejection corrects the decision and related charges. Public rules allocate and reimburse review costs, preventing unsupported repeated challenges from consuming other users' funds.

Failure and fraud are treated separately. When no node accepts a task, execution times out, verification is inconclusive, or data cannot be recovered, the protocol settles costs already incurred according to completed service and returns the refundable budget. Penalties require evidence of a specific violation. An inconclusive result caused by normal numerical variation is not itself cheating; unavailable data does not itself prove faulty model computation.

7. Batched On-Chain Submission and Blockchain Scaling

Each inference task generates records for participation, selection, output submission, verification, and settlement. These steps depend on one another: only the selected node may submit an execution result, and settlement cannot bypass the required verification conditions. Individually submitting each record and repeatedly reading and writing state becomes increasingly expensive as task volume grows.

TrueOpen uses the data availability (DA) network to retain complete task data. Builders collect stage records that the protocol permits to be combined and submit them on-chain in batches. The blockchain checks the records and processes payments under the task rules, amortizing submission and processing overhead across multiple tasks.

7.1 Complete Data in DA, Essential Records On-Chain

Inputs, inference outputs, and computation evidence are usually much larger than the records needed to confirm task progress. Repeatedly placing these materials in blocks would impose the corresponding transmission and storage costs on every consensus node.

The chain retains task identifiers, signatures, data commitments, and essential processing records; DA holds the complete underlying data. Verifiers and authorized reviewers retrieve it and check it against the commitments. This division lets the chain record what each participant submitted and the obligations they assumed, without storing all task content.

A commitment lets participants check whether materials have changed; it does not replace the materials themselves. DA must keep them available throughout verification and dispute resolution. An on-chain hash alone is not enough to conduct a review.

7.2 Batch Submissions Without Removing Per-Task Checks

Orders, participation requests, and signed records enter through the Canonical Inbox, which deduplicates and orders them under protocol rules and supports later submission of omitted records. BatchExecutor processes the ordered records. The main chain then verifies the results under the protocol before settling funds. A batch commitment binds the submitted data but does not, by itself, prove correct execution or authorize payment.

Builders group records from multiple tasks into batches where the protocol permits. Each record retains its task association, the submitter's signature, and the information needed to process it. The protocol checks each record's validity and updates the corresponding task state and balances.

Different verification methods produce different records. Logprob Verification by Majority Agreement must preserve the sequence of the commit, reveal, and decision phases; batching must not reveal committed values prematurely. Cryptographic proofs and numerical audits supply verification materials and results under their own rules. Combining submissions does not change these rules, nor does it automatically aggregate multiple proofs into one.

A single submission can thus carry records for multiple tasks while preserving independent checks and accountability for each task. A data commitment binds the submitter to the batch contents. It does not by itself prove that the records or computations are correct, and a commitment alone cannot justify releasing funds.

7.3 Higher Capacity with Bounded Waiting

Larger batches can reduce overhead per task, but waiting to fill them also increases latency. Batch processing must therefore bound both batch size and waiting time while respecting each task's existing deadlines. Inference outputs are still returned to users first, without waiting for batch submission or final settlement.

Batch processing must also support submission of omitted records and recovery from failures. If a Builder stops serving or omits a record, the task should not have to wait for that same node to recover. Resubmissions and retries must check the existing processing state to prevent a record from being executed twice or a fee from being paid twice.

Batching removes separate transactions, not the underlying signature verification, state access, or balance checks. Further gains can come from optimizing batch execution, caching, state writes, and indexing. Independent operations can also run in parallel where state dependencies permit. Block intervals and capacity must be tuned together. Shorter intervals reduce confirmation waits but neither remove verification steps nor guarantee higher task throughput.

Scaling should increase the number of verifications and settlements completed under sustained load while keeping latency and cost acceptable. Packing more records into a batch is not enough. On-chain processing, DA transfer, and recovery must scale together to avoid moving a backlog from one stage to another. Only workload tests can establish those gains; batch size alone cannot predict them.

8. Smart Contracts and AI Applications

Developers can use the network's inference services to build AI applications. Smart contracts manage fees, expenditures, and revenue distribution.

9. Identity Privacy and Optional Content Encryption

Account logins and credit-card payments can link service records to an email address, name, or payment identity. TrueOpen instead authenticates task authorization through blockchain addresses and digital signatures. Orders need not include a user's name, home address, or bank account. External information may still link an address to its owner, but real-world identity is not required for the protocol to accept a task.

Keeping identity details out of an order does not protect the submitted content. The network therefore offers optional encryption for inputs and inference outputs. When enabled, data is encrypted before upload; Builders store and relay ciphertext, while Workers and Verifiers receive only the decryption keys required for their task roles. The project team, storage nodes, and other organizations cannot decrypt user content merely by virtue of their identity or administrative privileges.

9.1 Delivering Encrypted Data to Selected Nodes

For each encrypted task, the SDK creates a fresh task master key and derives purpose-specific keys. It uploads an encrypted input, leaving candidate GPU nodes to apply using public task information alone. Once the Worker is selected using randomness from a future block that was unpredictable in advance, the SDK verifies both its on-chain assignment and the authenticity of its encryption public key before encrypting the master key for it. The Worker uses its private key to open the encrypted key package, then decrypts and checks the input before running inference.

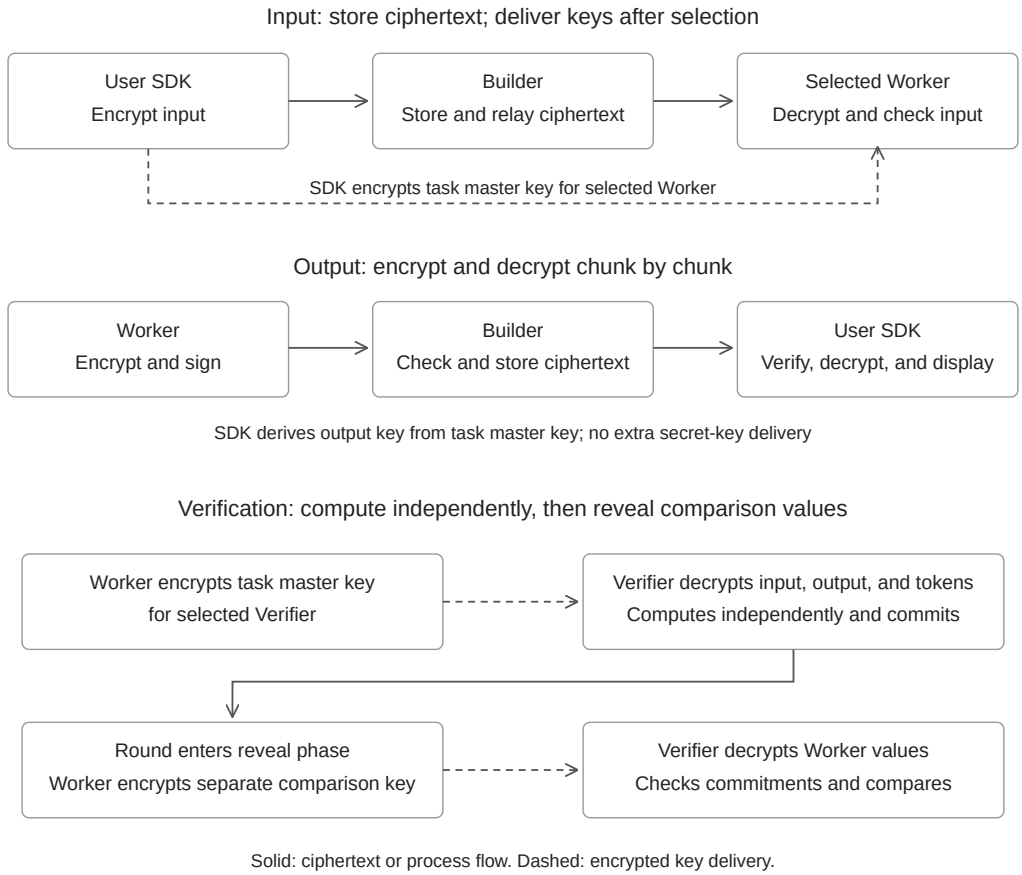
The scheme combines AES-256-GCM authenticated encryption for bulk data with ECIES encryption of the data key for designated recipients. Adding a recipient means producing another encrypted key package, not re-encrypting the full object. Encryption keys are managed separately from protocol signing keys, and the SDK rejects relay-supplied public keys whose identity has not been authenticated.

The output key is derived from the same task master key. The SDK already holds that key and can derive the output decryption key using the stream context signed by the Worker, without another secret-key delivery. Input, output, and original token records use purpose-specific subkeys. This separates uses, not access rights: anyone holding the master key can derive all of them.

9.2 Limiting Data Access to Authorized Verifiers

Once verifiers have been selected using future block randomness that was unpredictable in advance, the Worker checks their identities and task authorization before delivering the keys needed for verification. Applying as a candidate does not grant access to these keys. Builders likewise gain no decryption rights merely by storing and relaying ciphertext.

Authorized verifiers need to read the input, inference output, and original token sequence to check the computation independently. The Worker’s comparison values become accessible after verifiers have committed to their own values, subject to the reveal rules in Section 3.1.2. Encryption restricts who can obtain plaintext, but cannot prevent authorized nodes from disclosing content they have already read.



Builders may relay and retain encrypted key packages but cannot decrypt them. The task master key is independent of the key for the Worker's comparison values, which is delivered only in the reveal phase.

9.3 Checking Stored Data Without Decrypting It

Confidentiality does not prevent integrity checks. Producers separately commit to ciphertext and its underlying content, signing records that bind both to the task and delivery. Builders check received and stored bytes against ciphertext commitments; users and verifiers decrypt them and check the content commitments. Signing both commitments identifies who is accountable for the claimed correspondence, but recipients must still decrypt to verify that the two correspond.

Streamed outputs also require checks of chunk numbers, ordering, and final length. Builders verify cumulative ciphertext commitments; the SDK and Verifiers check cumulative plaintext commitments after decryption. AES-GCM authentication cannot replace the Worker's signature, because multiple authorized recipients may share the data encryption key.

Storage and relay thus need no decryption keys, but the scheme neither hides inputs from the Worker nor revokes plaintext already obtained by an authorized node. Encryption protects the content, but metadata such as data length and access times may remain public.

9.4 Online Obligations and Subsequent Review

Users choosing encryption must keep their SDK online until verification ends and supply required signatures and keys within the deadlines. If missing user authorization prevents the task or verification from continuing, the protocol still pays Worker and Verifier fees under the encrypted task's agreed terms. This condition is disclosed before submission. Payment does not itself establish that the computation passed verification.

Workers and Verifiers must also retain keys while their obligations remain open and deliver them to authorized recipients at the appropriate phase. When a new verifier joins a later review, the node responsible for key delivery must encrypt the key for that recipient. Existing encrypted key packages are usable only by their original recipients; a Builder cannot convert them for someone else. Even intact ciphertext cannot guarantee continued verification if every authorized key holder becomes unavailable.

9.5 Retention and Access Scope

Commitments and settlement records remain on-chain permanently. DA data must be retained until the obligations defined by the on-chain cleanup height and storage lease have ended, and cannot be deleted while verification or dispute obligations remain outstanding. The protocol does not let users unilaterally delete or withdraw previously submitted data.
Workers and Verifiers automatically delete locally retained plaintext inputs after a configured retention period, measured from completion of inference and verification, respectively. The retention period is determined by a protocol governance vote and enforced by the node software. This cleanup does not change the retention obligations for data stored in DA.

Participant	Accessible information
Public	Public on-chain authorization records, commitments, and settlement records; these do not grant access to task contents
Builder	Data it is responsible for storing and relaying; protected inputs, outputs, and evidence are ciphertext
Selected Worker	Plaintext inputs needed for inference and the outputs and evidence it generates
Selected Verifier	Data needed for verification; the independent key to the Worker's comparison values is released only in the reveal phase
Authorized reviewer	Data within the dispute's scope; initiating a storage challenge does not automatically grant decryption rights

10. Conclusion

Open-source models allow people to run AI themselves. But for GPUs from different providers to serve users, the network must also address trust in the computation, payment, and the privacy of user content. With TrueOpen, users do not have to rely solely on a provider’s word that the computation was performed. Payments are settled based on verification results, and nodes that violate the rules are held accountable.

Users receive inference results without waiting for verification and settlement to finish. They can also protect the privacy of their inputs and outputs through content encryption and controlled access to decryption keys. Different providers can contribute compute to the same network, and developers can use it to build AI services without a single platform controlling both the services and user data.

References

[1] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008.
[2] TrueOpen. SLP Verifiable Inference: Proof-of-Concept Experiment Report (with raw run logs and data tables). September 2026. Report: https://github.com/TrueOpen/slp-experiments/blob/main/REPORT.md ; data repository: https://github.com/TrueOpen/slp-experiments

[3] TrueOpen. Logprob Majority-Consensus Verification (LMCV): A Low-Cost Execution-Verification Protocol for Open Inference Networks. September 2026. Paper: https://github.com/TrueOpen/lmcv-experiments/blob/main/paper/LMCV.md
[4] TrueOpen. Qwen3-8B BF16 Single-Sample Verifier Design (with raw run logs and data tables). September 2026. Report: https://github.com/TrueOpen/lmcv-experiments/blob/main/experiments/logprobs/reports/LLM-QWen3_8b_bf16_single_sample_verifier_params.md
[5] TrueOpen. Qwen3.6-35B-A3B 6000ws Worker-Verifier Logprob Comparison Experiment Report (with raw run logs and data tables). September 2026. Report: https://github.com/TrueOpen/lmcv-experiments/blob/main/experiments/logprobs/reports/qwen3_6_35b_a3b_6000ws_verifier_testing_report.md
[6] TrueOpen. Worker vs. Verifier Latency Benchmark Report (with raw run logs and data tables). September 2026. Report: https://github.com/TrueOpen/lmcv-experiments/blob/main/experiments/worker-verifier-latency/reports/worker_verifier_latency_benchmark_report.md
[7] TrueOpen. MoE Routed Experts Verifier Design (with raw run logs and data tables). September 2026. Report: https://github.com/TrueOpen/lmcv-experiments/blob/main/experiments/moe-routed-experts/report/moe_model_routed_experts_verifier.md
[8] Sun, H., Li, J., & Zhang, H. (2024). zkLLM: Zero knowledge proofs for large language models. ACM CCS 2024. arXiv:2404.16109
[9] Wang, Z. (2026). NanoZK: Privacy-preserving verifiable inference for large language models via layerwise zero-knowledge proofs. arXiv:2603.18046
[10] Sanjaya, P. K., Giannoula, C., Oktavian, V., Saeedi, M., Sines, G., Saileshwar, G., & Vijaykumar, N. (2026). zkComposer: Decomposing proof construction to scale zkML. arXiv:2607.08095
[11] Baser, O., Sadeghi, E., Wang, E., Alves, D. R., Kazemian, S., Kang, H., Chinchali, S. P., & Vishwanath, S. (2026). TensorCommitments: A lightweight verifiable inference for language models. arXiv:2602.12630
[12] Yao, J., Su, H., Liao, T., Cheng, Z., Zhang, H., Wang, X., & Viswanath, P. (2026). TAO: Tolerance-aware optimistic verification for floating-point neural networks. EuroSys 2026, pp. 1515–1532. doi:10.1145/3767295.3803612
[13] Zamir, O. (2026). A note on non-composability of layerwise approximate verification for neural inference. arXiv:2602.15756
[14] Gong, C., Liu, B., & Li, M. (2026). Hollow-LLM attack: Computationally trivial weights in zero-knowledge verification of LLM inference. IEEE Symposium on Security and Privacy 2026. doi:10.1109/SP63933.2026.00258
[15] Guo, Y., Qu, W., Wu, L., Zhai, S., Wang, L. Z., Xu, M., Liu, Y., Yuan, B., Song, D., & Zhang, J. (2026). IMMACULATE: A practical LLM auditing framework via verifiable computation. arXiv:2602.22700
[16] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating Computation: Interactive Proofs for Muggles. ACM STOC, 2008.
[17] Cosmos SDK. Cosmos SDK Architecture. Cosmos SDK v0.53 documentation.
[18] Rūsiņš Freivalds. Fast Probabilistic Algorithms. MFCS 1979, Lecture Notes in Computer Science, vol. 74, pp. 57–69, 1979.
[19] Eigen Labs. EigenDA: The Hyperscale Verifiable Data Availability Layer.
[20] Celestia. Data Availability. Official documentation.
[21] Avail. How Avail DA Works. Official documentation.