ICJK
Journal of Reliable and Secure Computing
http://dx.doi.org/10.62762/JRSC.2026.867713

RESEARCH ARTICLE

TrustCompute: Result Validation, Reputation-Aware Scheduling and Quality-Based Payments for Outsourced LoRA Fine-Tuning

Lin-Fa Lee[1], **Yi-Yu Chang**[1] **and Kuo-Hui Yeh**[1,*]

[1] Institute of Artificial Intelligence Innovation, National Yang Ming Chiao Tung University, Hsinchu 300093, Taiwan

Abstract

Outsourced model fine-tuning requires reliable task allocation, result validation, and incentives for quality. We present TrustCompute, a compute-sharing platform integrating two-stage adapter verification, reputation-aware scheduling, quality-based payments, and auditable ledger records for LoRA fine-tuning. Our evaluation combines synthetic workloads, real GPU-based fine-tuning, and a local Ethereum Virtual Machine (EVM) environment. Across 440 validation cases, all 400 invalid submissions were rejected and all 40 valid submissions were accepted. With 50% unreliable workers, reputation-aware scheduling achieved 100% task completion and reduced the mean failed-attempt time by 89% relative to round-robin scheduling. An integration experiment completed 120 real fine-tuning tasks. Additional simulations showed that identity resets can exploit exploration and that quality proportional payments can improve strategic workers' delivered quality. These findings support jointly designing verification, scheduling, and payment mechanisms. TrustCompute verifies compliance with result acceptance criteria, but does not prove adherence to a prescribed training procedure.

Keywords: outsourced computing, LoRA fine-tuning, result validation, reputation-aware scheduling, quality-based payment, auditable ledger.

1 Introduction

Parameter efficient adaptation makes it possible to specialize a pretrained model while exchanging a relatively small update rather than an entire model. Low-rank adaptation (LoRA) freezes the base weights and trains low-rank matrices attached to selected modules [1]. These developments motivate a practical outsourcing unit: a requester specifies a fine-tuning job and an external provider returns an adapter that can be loaded and evaluated independently of the provider's infrastructure. Ensuring reliable participation and quality-aware payments in such a setting requires explicit mechanism design [2].

Shared computing has a longer history. The Berkeley Open Infrastructure for Network Computing (BOINC)

Submitted: 09 July 2026
Revised: 14 September 2026
Accepted: 16 September 2026
Published: 19 September 2026

Vol. 2, **No.** 3, 2026.
10.62762/JRSC.2026.867713

***Corresponding author:**
✉ Kuo-Hui Yeh
khyeh@nycu.edu.tw

Citation
Lee, L. F., Chang, Y. Y., & Yeh, K. H. (2026). TrustCompute: Result Validation, Reputation-Aware Scheduling and Quality-Based Payments for Outsourced LoRA Fine-Tuning. *Journal of Reliable and Secure Computing*, 2(3), 179–193.

organizes public resources around distributed jobs and returned results [3]. More recent systems, including Petals [4] and Learning@home [5], explore collaborative model execution and training across participants. Such systems make resource coordination an important part of machine learning infrastructure. Yet access to nominal computing capacity does not by itself establish that an assigned job will finish on time or produce an acceptable model update.

We consider a platform that outsources individual LoRA jobs to partially trusted providers. A provider may be slow, disappear after accepting work, exaggerate throughput, or submit a malformed adapter. It may also return an untrained or low-quality adapter whose files appear structurally correct. Even a provider that always passes validation can behave strategically: when every accepted result receives the same payment, additional quality may have no immediate monetary value to that provider.

These behaviors create separate requirements. The platform must validate the current result, decide which provider should receive future work, and define what quality the payment rule rewards. A hash commitment addresses whether bytes match a recorded commitment, not whether those bytes implement a useful update. Reputation summarizes observed history, but cannot replace evaluation of a new result. A failure penalty penalizes rejected work but does not necessarily incentivize higher-quality outputs once the acceptance threshold has been met.

TrustCompute connects these requirements through an explicit task lifecycle. The requester commits a specification, the scheduler selects an eligible worker, the worker submits an adapter, and the validator checks structure followed by held-out quality. The coordinator records the outcome, updates reputation, and either completes the job or schedules another attempt. A replaceable ledger records task commitments, assignments, validation outcomes, and reputation changes. An experimental economic layer evaluates flat and quality proportional payment rules under a specified effort model.

The study asks four questions. RQ1 asks whether two-stage validation distinguishes constructed structural faults from quality faults. RQ2 asks how scheduling policies affect completion, retries, and failed attempt time as unreliable participation increases. RQ3 asks whether the complete workflow operates with real fine-tuning and local EVM recording, and how its management cost compares with job duration. RQ4 asks how conditional defection, identity resets, and effort sensitive payments affect behavior beyond static faults.

Our contribution is an implemented integration and a controlled evaluation, rather than a new cryptographic proof system or a claim of optimal scheduling. First, we provide a common execution path for validation, retries, reputation updates, and ledger events. Second, we evaluate that path using both a synthetic adapter backend and real LoRA artifacts. Third, we identify conditions under which exploration and flat payments produce undesirable assignment or quality outcomes. The resulting design guidance is conditional on the tested workloads and explicit trust assumptions.

2 Related Work

2.1 Shared resources and collaborative learning

BOINC [3] provides infrastructure for public resource computing, including mechanisms for distributing work and managing returned results. Its relevance is the system level need to coordinate contributors whose availability and hardware differ. TrustCompute adopts an application specific task contract: the returned artifact is an adapter, and acceptance depends on compatibility and a requester defined quality threshold.

Petals [4] distributes model execution across participants and exposes interfaces for inference and adaptation of large models. Learning@home [5] studies crowdsourced neural network training using decentralized experts and poorly connected contributors. These approaches address how participants jointly execute a large model. Our prototype instead assigns each adapter job to one logical worker at a time and retries the whole job when an attempt fails. We do not implement model partitioning or claim the communication performance of either collaborative system.

Weng et al. [6] propose DeepChain, a distributed framework that combines blockchain-based incentives with auditable gradient collection to enforce correct behavior among participants. Its value-driven incentive mechanism and ledger-backed traceability address a concern similar to ours: that participants may act incorrectly without external accountability. TrustCompute shares the goal of auditable task records and quality-linked rewards, but applies these to single-provider adapter delivery rather than multi-party gradient aggregation.

2.2 Verifying outsourced computation and training

Pinocchio [7] represents a cryptographic approach to checking outsourced computation: the worker supplies a proof for a specified computation, which is checked using a verification key. Slalom [8] combines trusted execution with delegated neural network operations and evaluates verifiable inference. These works illustrate stronger execution related assurances and different implementation assumptions than empirical assessment of a final adapter.

VeriML [9] is particularly relevant because it addresses outsourced machine learning correctness, resource accounting, and fair payment. Its proposed design uses proofs on sampled training iterations to provide probabilistic integrity assurance. Proof-of-Learning [10] likewise concerns evidence about the training process rather than only the final model's predictive quality. TrustCompute does not verify individual optimization steps, produce training proofs, or ensure that claimed resource consumption equals actual hardware use.

Körbel et al. [11] use blockchain based verification of off-chain results with zero knowledge proofs. Our ledger integration has a narrower function. A trusted validator computes an off-chain verdict and the coordinator records it. The contract does not independently establish that the adapter was trained correctly. Consequently, we compare the assurance targets conceptually and do not report runtime superiority over proof based systems that solve a stronger problem.

2.3 Reputation and exploration

EigenTrust [12] aggregates peer interaction history into trust values used to select sources. The Beta Reputation System [13] represents positive and negative evidence with a beta distribution. TrustCompute uses directly observed acceptance and failure events instead of third-party ratings. Its primary scalar rule rewards validated quality gradually and penalizes unsuccessful attempts more sharply. A second backend maintains quality weighted beta pseudo counts.

Upper Confidence Bound (UCB) methods [14] and Thompson sampling [15] offer ways to balance exploitation with exploration. New providers need opportunities to establish a history, but exploration exposes the platform to uncertain outcomes. We evaluate a UCB style index and posterior sampling inside the same eligibility and queueing framework. The quality weighting, discounting, changing candidate set, and nonstationary worker behavior differ from a standard stationary bandit model. Classical regret bounds therefore do not automatically apply to these implementations.

2.4 Identity and payment incentives

Friedman and Resnick [16] analyze how cheap pseudonyms allow participants to escape reputational consequences and discuss the trade-offs of entry costs. Douceur [17] explains the broader identity problem behind Sybil attacks, in which one entity can present multiple identities. Our whitewashing experiment models a restricted consequence of cheap identity: resetting a worker's reputation to the newcomer state. It does not implement or defeat a general Sybil adversary with an unbounded population of simultaneous identities.

Incentive design for learning must account for the costs borne by participants. Ding et al. [18] design optimal contracts for federated learning under multi-dimensional private information, demonstrating how payment structure shapes workers' effort and quality disclosure. Our experiment concerns a different decision: how much effort one provider invests in a stand-alone job when payment depends on its accepted quality. We compare two simple payment rules under a fixed production curve and cost model, without claiming incentive compatibility, an equilibrium for an open market, or truthful resource reporting.

2.5 Position of this study

Existing work motivates each component, but the present evaluation asks how the components interact in one adapter delivery workflow. The empirical distinction is consequential. An invalid result can be rejected while still consuming assignment opportunities. A valid result can meet a minimum threshold while offering less quality than the requester would buy under another payment rule. An auditable record can preserve a verdict without eliminating the need to trust its evaluator. These distinctions determine both the platform design and the metrics used below.

3 System Model and Design

3.1 Roles and trust assumptions

The requester defines the task and acceptance policy. The coordinator manages eligibility, assignments, state transitions, and retries. Workers execute training and submit adapters. The validator checks artifacts and

measures quality. The ledger records the lifecycle. Figure 1 summarizes the feedback path from validation to future assignments and accounting.

The requester, coordinator, and validator are trusted. Workers are partially trusted and their advertised capabilities can be inaccurate. The ledger is assumed to execute its rules correctly. In the EVM implementation, a designated owner submits contract operations. This is an application controlled by a trusted coordinator with ledger backed records, not a permissionless marketplace with decentralized quality judgment.

Workers receive a training view or training split, while the validator evaluates a separate held-out view or split. This describes the experimental data path. For the real experiment, the dataset is public; process level separation is not a confidentiality guarantee against an external worker that can download the same data. Privacy, collusion, model exfiltration, and a fully adversarial requester are outside the threat model.

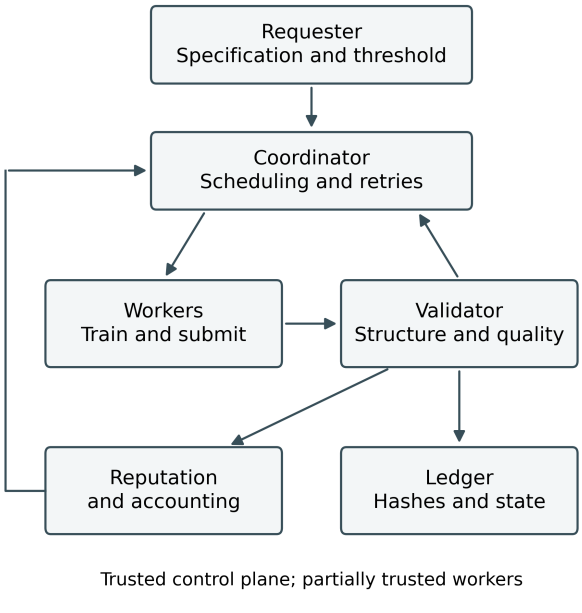

Figure 1. TrustCompute workflow. The coordinator routes jobs and handles retries. Validation outcomes feed reputation and accounting; the ledger records commitments and state. Training artifacts remain off-chain.

3.2 Job specification and state management

A job identifies the base model and revision, dataset and revision, LoRA configuration, training settings, required memory, quality threshold, timeout, and maximum attempts. For real adapters, the requester also supplies a tensor schema derived from the expected adapter layout. Binding these items to a configuration hash makes the task specification uniquely identifiable throughout the lifecycle, while leaving large objects outside the ledger.

The coordinator creates an attempt only after selecting an eligible worker. A submitted adapter is associated with the job, attempt number, worker, artifact path, and artifact hash. It passes through validation before the coordinator accepts or rejects the attempt. Acceptance completes the job. Rejection or timeout can trigger reassignment until the attempt budget is exhausted. A successful final status therefore does not erase earlier failures.

Candidate eligibility depends on worker availability, advertised memory, and the bounded backlog. The event driven simulator uses a default capacity of one running job per worker and a maximum queue depth of three. When no worker can admit another assignment, work waits in a global queue. These admission rules are shared across policies to avoid comparing unrestricted queue accumulation with bounded admission.

Timeouts begin when execution starts rather than when work is first assigned. Queue waiting contributes to time to result, but does not directly penalize a worker's reputation. A dropout is charged the full deadline in the failed attempt metric because the platform learns of the missing result at timeout. This accounting convention is conservative about platform waiting, and is not a measurement of device energy consumption.

3.3 Structural validation

LoRA represents an update to a frozen weight matrix using low-rank factors [1]. For a base matrix W, the adapted weight is:

$$W' = W + \frac{\alpha}{r} BA \tag{1}$$

Here r is the rank, α is a scale parameter, and A and B are trainable matrices. Structural compatibility is necessary because the shapes and names of these tensors must match the requested target modules. Compatibility does not establish that the tensors encode useful training.

Stage 1 uses fail-fast checks to verify artifact existence, hash consistency, and configuration readability. It then checks the declared base model and revision, LoRA rank and scaling settings, and target modules against the job specification. Finally, it verifies that the tensors can be loaded, match the expected names

and shapes, and contain only finite values. The real backend uses an explicit per-tensor schema because grouped query attention can give different projection modules different shapes. Uniform shape assumptions used by a mock backend cannot safely be transferred to a real model.

A failed check yields a rejection without a held-out model evaluation. This separates inexpensive structural failures from artifacts that require behavioral assessment. Metadata matching has a limited meaning: a worker may write the expected revision string without having trained that revision. The implementation checks consistency of declarations, not the provenance of the entire training process. Similarly, matching a submitted hash ensures consistency with that commitment; it does not make maliciously chosen bytes benign.

3.4 Quality validation

Stage 2 evaluates only structurally compatible adapters. For a higher is better metric, quality is normalized against a base value and a target value:

$$Q = \mathrm{clip}\left(\frac{M_{\mathrm{obs}} - M_{\mathrm{base}}}{M_{\mathrm{target}} - M_{\mathrm{base}}}, 0, 1\right) \tag{2}$$

A submission is accepted if Stage 1 passes and Q is at least the task threshold τ. The target fixes the point receiving full normalized credit, while τ fixes the minimum accepted fraction of that improvement. These two quantities must not be confused.

In the synthetic backend, the task defines a hidden low-rank direction. The worker has a noisy training view and the evaluator uses an independently noisy held-out view. The evaluator measures alignment of the submitted update with the held-out direction. An untrained adapter has zero update under the initialized LoRA factors; random factors generally do not recover the task direction. This backend provides a controllable quality task for exercising the control plane without a GPU.

For the real backend, the metric is loss improvement over the base model on the validation split:

$$Q = \mathrm{clip}\left(\frac{L_{\mathrm{base}} - L_{\mathrm{adapter}}}{\Delta L_{\mathrm{target}}}, 0, 1\right) \tag{3}$$

The experiment uses a target improvement of 0.10 nats per token and $\tau = 0.5$. Thus, passing requires an improvement of at least 0.05 nats per token on the evaluated blocks. The normalized score saturates at one. A score near one is not a classification accuracy, does not quantify general capability, and does not imply that all accepted models perform identically beyond the saturation point.

3.5 Reputation updates

The primary backend initializes each worker at R = 0.5. A successful attempt updates the scalar reputation using validated quality:

$$R' = R + \eta_+(1 - R)Q \tag{4}$$

An unsuccessful attempt applies:

$$R' = (1 - \eta_-)R \tag{5}$$

We use $\eta_+ = 0.1$ and $\eta_- = 0.4$. Trust consequently grows through successful observations but falls after rejection, timeout, or abandonment. The parameters are design settings, not learned optima. Slow and capability misreporting workers can both appear as timeouts, so the rule estimates observed reliability rather than diagnosing the underlying cause.

The alternative backend initializes beta pseudo counts a = b = 1. At each observed outcome, both counts are multiplied by λ, subject to a small numerical floor. Acceptance adds Q to a; failure adds one to b. The point estimate is a/(a + b). We evaluate $\lambda = 1$ and a supplementary discount setting $\lambda = 0.95$. Discounting occurs on updates, not continuously with wall-clock time.

Quality enters as fractional positive evidence. This is a beta shaped reputation heuristic rather than an exact Bernoulli posterior for independently observed binary successes. In particular, lower accepted quality slows the accumulation of positive evidence; it does not add the complementary quantity 1 - Q as failure evidence. This detail affects both the score and the uncertainty used by the alternative schedulers.

3.6 Scheduling policies

The primary utility for eligible worker i is:

$$U_i = 0.5R_i + 0.3H_i + 0.1\mathrm{Av}_i - 0.1L_i \tag{6}$$

R is reputation, H is hardware compatibility, Av is availability, and L is normalized estimated latency. If the worker has queue length q, Av = 1/(1 + q).

Estimated latency is calculated by dividing nominal training time by declared throughput and multiplying the result by 1 + q. This estimate is then normalized by the largest estimated latency among the currently eligible candidates. Hence the score depends on both the worker and the current queue state.

The hardware score clips declared memory divided by requested memory to [0, 1]. Because eligibility already requires sufficient declared memory, this term is one for eligible candidates in the present implementation. It does not provide a graded preference among those candidates. The reported experiments must therefore not be interpreted as demonstrating sophisticated hardware aware matching.

With probability $\epsilon = 0.1$, the reputation-aware scheduler selects an eligible worker uniformly at random. Otherwise, it selects the worker with the highest utility. Ties are resolved deterministically using worker identifiers. Round robin cycles through eligible workers. The resource only policy uses 0.6H + 0.2Av - 0.2L and no history. Since weights on non-reputation terms also differ, comparisons between these policies measure complete policy configurations rather than the isolated causal effect of adding reputation.

The Thompson variant substitutes a draw from each worker's beta distribution for R in Equation (6). The UCB variant substitutes:

$$I_i = \frac{a_i}{a_i + b_i} + c\sqrt{\frac{2\ln(\max(t, 2))}{\max(n_i, 1)}} \tag{7}$$

Here t counts scheduling decisions and n is the pseudo count mass beyond the prior. The numerical denominator is floored at one. The index is not clipped to [0, 1], because it is a ranking score rather than a probability. With discounting and fractional quality updates, n is not the literal number of completed jobs. We compare c = 0.10, 0.25, and 0.50 rather than treating the exploration scale as universal.

3.7 Ledger and off-chain records

The ledger interface supports worker registration, job commitment, assignment, result commitment, validation recording, reputation updates, and job finalization. The memory and EVM backends expose corresponding operations. The Solidity contract stores compact identifiers, hashes, verdicts, and integer quality and reputation values; adapters and full configuration objects remain off-chain.

The EVM backend uses eth-tester and py-evm in process. It checks the receipt status after sending a transaction so that a mined revert is not reported as a successful write. The repository also includes equivalence tests that drive both backends through a common operation sequence and compare normalized state. These tests support interface consistency, but do not establish equivalence under every possible workload or adversarial contract interaction.

The owner authorized contract trusts the submitted quality verdict. Ledger recording therefore improves traceability of the coordinator's decisions under the stated trust assumptions. It neither removes the trusted validator nor supplies the proofs used by systems such as Pinocchio and blockchain based result verification [7, 11]. No real currency transfer, stake custody, or permissionless settlement is evaluated.

3.8 Payment and effort model

Let w be the fee for a full credit job, e in [0, 1] the effort fraction, k the full effort computing cost, and s a failure penalty. A flat rule pays w for any accepted result. A proportional rule pays wQ for an accepted result. Rejected results receive no fee under either rule. The modeled per-attempt utility is:

$$\pi(e) = P(Q(e)) - ke - s\mathbf{1}[Q(e) < \tau] \tag{8}$$

The simulator represents $Q(e)$ using a fixed 21 point lookup table. Effort values range from 0 to 1 in increments of 0.05. Values between adjacent points are obtained by linear interpolation. The table is based on synthetic-backend calibration and is treated as a simulation assumption rather than an empirical relationship for real LoRA training. The code searches the finite effort grid {0, 0.05, ..., 1.00} for the highest utility. We do not assume global concavity or use a derivative based optimum for this tabulated function. At the main parameters w = 1, k = 0.3, and $\tau = 0.5$, the selected grid effort is 0.40 under flat payment and 0.70 under proportional payment.

The platform also records an entry cost each time an identity is registered or reset. The registration cost is deducted permanently whenever an identity is registered or reset, with no refund mechanism. The account's profit equals fees minus compute costs, penalties, and registration costs. Penalties and registration costs are accounting quantities, not measurements of enforceable real world losses.

The shirker changes effort according to the modeled

Table 1. Overview of the evaluation experiments. Each simulation run uses a specified configuration and random seed. Main and supplementary settings are reported separately.

Study	Scope / configuration	Evaluation size
E1	11 submission categories	330 mock + 110 real cases
E2	3 policies × 6 fractions × 10 seeds	180 runs; 36,000 jobs
E3	Real LoRA and local EVM	120 jobs; 127 attempts
E4	7 operations × 2 backends	200 repetitions per operation–backend pair
E5	4 arms × 6 fractions × 10 seeds	240 runs; 48,000 jobs
E6	6 mechanisms × 5 scenarios × 4 fractions × 10 seeds	1,200 runs; 240,000 jobs
E7	2 payment rules × 4 penalties × 4 scenarios × 2 fractions × 10 seeds	640 runs; 128,000 jobs
E6	Evidence discount $\lambda = 0.95$	1,200 runs; 240,000 jobs
E7	Registration cost 0.5	640 runs; 128,000 jobs
E7	Compute cost 0.15	80 runs; 16,000 jobs
E7	Compute cost 0.60	80 runs; 16,000 jobs

best response. Lazy and whitewashing behaviors remain programmed behaviors even when their recorded profit becomes negative. The experiments can therefore show changes in modeled profitability, but cannot demonstrate that a loss making attacker leaves the platform. Participation constraints, credit limits, and endogenous entry are not implemented.

4 Experimental Method

4.1 System Implementation

TrustCompute is implemented in Python and includes a task coordinator, schedulers, two-stage result validators, exponential-moving-average (EMA) and beta reputation backends, an economic accounting module, and a discrete-event simulator. The platform provides a FastAPI interface for task operations and supports execution backends for synthetic workloads and real LoRA fine-tuning. The ledger supports both in-memory and local EVM implementations. Solidity smart contracts record the task lifecycle, with contract operations handled through `eth-tester`, `py-evm`, and `web3`.

The experimental environment uses Linux and Python 3.12.3. The compute node is equipped with NVIDIA H100 80GB HBM3 GPUs, two Intel Xeon Platinum 8480C processors, and approximately 2 TB of system memory. Each execution job for E1-real and E3 is allocated one H100 GPU and eight CPU cores, while E4 is allocated eight CPU cores. The model training environment uses PyTorch 2.11.0+cu128, CUDA 12.8, Transformers 5.16.1, PEFT 0.20.0, and `datasets` 5.0.1. The local ledger uses `eth-tester` 0.14.0b1, `web3` 8.0.0, and Solidity compiler version 0.8.26.

The real fine-tuning workload uses Qwen2.5-0.5B [19] and the `wikitext-2-raw-v1` configuration of WikiText-2 [20]. Training and validation data are drawn from the train and validation splits, respectively, using 512 training blocks and 64 validation blocks, each containing 256 tokens. LoRA is applied to the `q_proj` and `v_proj` modules with rank 8, a scaling parameter of 16, and zero dropout. Each fine-tuning task performs 30 optimizer steps with a learning rate of 0.0002 and a batch size of four.

4.2 Experiment matrix

Table 1 summarizes the scope and size of experiments E1–E7, including the supplementary settings for E6 and E7.

Table 1 distinguishes the principal grids from their supplementary variants. In particular, the main E7 grid contains 640 runs and 128,000 jobs; including its registration-cost and computing-cost variants gives 1,440 runs and 288,000 jobs.

E1 constructs clean, low quality, untrained, random weight, non-finite, shape mismatched, missing tensor, wrong revision, wrong rank, truncated file, and hash mismatched submissions. The synthetic backend uses 30 cases per category; the real backend uses ten. These are controlled tests of specified faults, not a sample from the distribution of all malicious adapters.

E2 uses ten workers, 200 jobs per run, and unreliable fractions of 0, 0.1, 0.2, 0.3, 0.4, and 0.5. Ten seeds are used at each policy and fraction. The runner specifies an arrival rate of 0.12 jobs per simulated second. The unreliable mix includes lazy, dropout, corrupt, slow, and capability misreporting behavior. As the unreliable fraction changes, the finite worker roster also changes its fault composition, so the sweep is not a pure fraction change at a fixed infinitely divisible mixture.

E3 fixes ten logical workers: five honest, one slow, one

dropout, one lazy, one corrupt, and one misreporting. The E3 execution summary records this fixed roster, and all ten worker identifiers appear in the attempt records. The deadline is twice a warm honest training measurement: 2.484094 seconds for a calibrated honest duration of 1.242047 seconds. The maximum attempt count is three. The run uses real training but advances a virtual clock with measured durations; slow and misreporting behaviors scale those durations.

E4 repeats worker registration, job commitment, assignment, result commitment, validation recording, reputation updating, and finalization 200 times on each backend. E5 crosses round robin or reputation-aware scheduling with normal validation or an acceptance override. The override still performs some structural check work before forcing the verdict, and is not a zero validation cost implementation.

E6 compares persistent laziness, two conditional defection settings, whitewashing, and mixed adaptive behavior at unreliable fractions 0.1, 0.2, 0.3, and 0.5. Its six configurations are round robin, scalar reputation with ϵ-greedy exploration, beta reputation with Thompson sampling, and three UCB coefficients. E7 compares shirker, lazy, whitewashing, and mixed scenarios at fractions 0.2 and 0.4. It uses penalties 0, 0.25, 0.5, and 1.0 under both payment rules.

4.3 Metrics and denominators

Job completion is the number of jobs ending with an accepted result divided by submitted jobs. When validation is disabled, this is only a state based completion measure: an accepted artifact may be invalid. Invalid acceptance is the number of constructed invalid submissions accepted divided by invalid submissions actually reaching validation. Valid rejection uses valid submitted artifacts as its denominator. A timed out attempt with no validation report is not an invalid submission in these ratios.

Honest selection counts assignments to the honest worker class among all attempts. Whitewasher or adversary share likewise uses attempts, not the worker population. In conditional defection scenarios, adversarial class selection need not be a defective submission: validity must be determined for each attempt using its planned behavior. These denominators prevent a high assignment share from being mistaken for successful malicious result acceptance.

Failed attempt time sums the platform-recorded time of attempts that do not produce an accepted adapter. Dropouts incur the deadline charge described in Section 3.2. Mean time to an accepted result is conditional on successful jobs and must be interpreted with completion. Retry count is the number of additional attempts per job. Control plane overhead is control plane time divided by recorded computation time, rather than by total elapsed time.

For economic evaluation, we distinguish quality among accepted shirker submissions from quality among all accepted submissions. Profits averaged across workers and profits pooled per attempt use different weights. For each E7 run, quality is averaged over accepted submissions within the relevant worker group. The reported quality is then the mean of these run averages across ten seeds, with each run receiving equal weight.

4.4 Evidence handling and statistical interpretation

Numerical results are derived from recorded case, attempt, and run summaries. Consistency checks use the underlying JSONL events where the required quantities are available, and processed records otherwise. Event files containing multiple executions are segmented at `run.start` records to prevent aggregation across separate executions. In the checked E2, E6-main, and E7-main records, final-segment job counts agree with the corresponding processed tables for all 2,020 analyzed runs.

The main E6 analysis contains 1,200 runs of 200 jobs each. Ten additional preliminary runs, each configured for 60 jobs under an earlier UCB implementation, are excluded based on configuration and experimental purpose rather than outcome. These preliminary runs are absent from the main E6 results table. Main and supplementary experiments are analyzed separately according to Table 1.

Figures 3, 6, and 7 use run level means and population standard deviations across the ten recorded seeds. Shaded bands or error bars describe this between run spread and are not confidence intervals for a general deployment population. Tables reporting pooled proportions are explicitly identified. Shared reputation histories make jobs within a run dependent; the job count is not an equivalent number of independent replications. No significance claim or optimality claim is made from the pooled counts.

5 Results

5.1 E1 validation outcomes

Across the 440 constructed cases, all 400 invalid submissions were rejected and all 40 valid submissions were accepted. Stage 1 rejected 280 structural faults and Stage 2 rejected 120 quality faults. Table 2 gives the denominators by backend. Figure 2 shows the real adapter categories, for which the structural versus behavioral distinction is especially important.

Table 2. Validation counts. IA denotes invalid artifacts accepted and VR denotes valid artifacts rejected. Stage counts include invalid cases only.

Backend	Valid / invalid	Stage 1 / 2	IA / VR
Synthetic	30 / 300	210 / 90	0 / 0
Real LoRA	10 / 100	70 / 30	0 / 0
Total	40 / 400	280 / 120	0 / 0

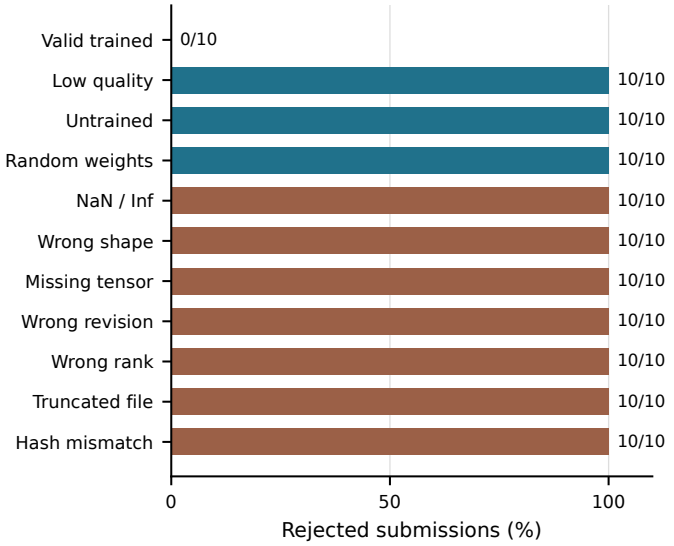

Figure 2. E1 real-adapter validation outcomes by submission category.

The random-weight and untrained cases passed far enough through the structure path to require quality assessment. Conversely, a truncated file or a tensor-name mismatch could be rejected before running held-out inference. This division supports the two-stage design: compatible serialization is necessary but insufficient for a useful adapter.

The real-adapter test has only ten clean examples and 100 invalid examples. The nominal 95% Wilson upper bound for invalid acceptance is 3.70% despite observing zero accepts. The category construction and repeated seeds also limit generalization beyond the finite test set. We therefore report observed separation of these faults, not a universal false-accept probability of zero.

5.2 E2 scheduling with unreliable workers

At 50% unreliable participation, reputation-aware scheduling completed all 2,000 jobs across ten runs. Round robin completed 1,867 and resource-only scheduling completed 1,803. Table 3 reports the pooled completion and honest-selection proportions alongside averages of the per-run continuous metrics.

Table 3. E2 at 50% unreliable workers. RR is round robin, RO is resource only, and RA is reputation-aware. Each policy has ten runs of 200 jobs. Time-to-result averages include successful jobs only; failed time is per run.

Metric	RR	RO	RA
Completion (%)	93.35	90.15	100.00
Honest attempts (%)	55.38	50.08	93.07
Retries per job	0.6855	0.8000	0.0745
Time to result (s)	67.77	42.93	31.44
Failed time (s/run)	6,121.66	5,827.05	672.47

Relative to round robin, reputation-aware scheduling reduced mean failed attempt time by 89.0%. Relative to resource-only scheduling, the reduction was 88.5%. The associated decrease in retries and increase in honest assignments are consistent with avoiding providers that repeatedly fail observed tasks. The resource-only policy can continue favoring a provider whose advertised speed is attractive even when that advertisement does not predict successful delivery.

This advantage is conditional. In the all-honest setting, resource-only scheduling reached accepted results in 21.52 seconds on average, compared with 24.56 seconds for reputation-aware scheduling and 24.94 seconds for round robin. The history-sensitive policy is not uniformly fastest. Its exploration and preference for established providers can be unnecessary when all available providers are already reliable.

Figure 3 shows the full fraction sweep. Between-run variability is particularly visible in the resource-only failed-time measurements as faults become more prevalent. A policy that looks competitive on conditional completion time can still lose jobs or consume substantially more failed attempt time. The combination of metrics is therefore more informative than a single latency ranking.

The comparison establishes the behavior of the configured policies. It does not isolate reputation from the changed availability and latency weights. A matched-weight ablation would be needed before

attributing the entire improvement to Equation (4). Nor does the clipped hardware term establish a benefit from selecting finer-grained hardware configurations among eligible workers.

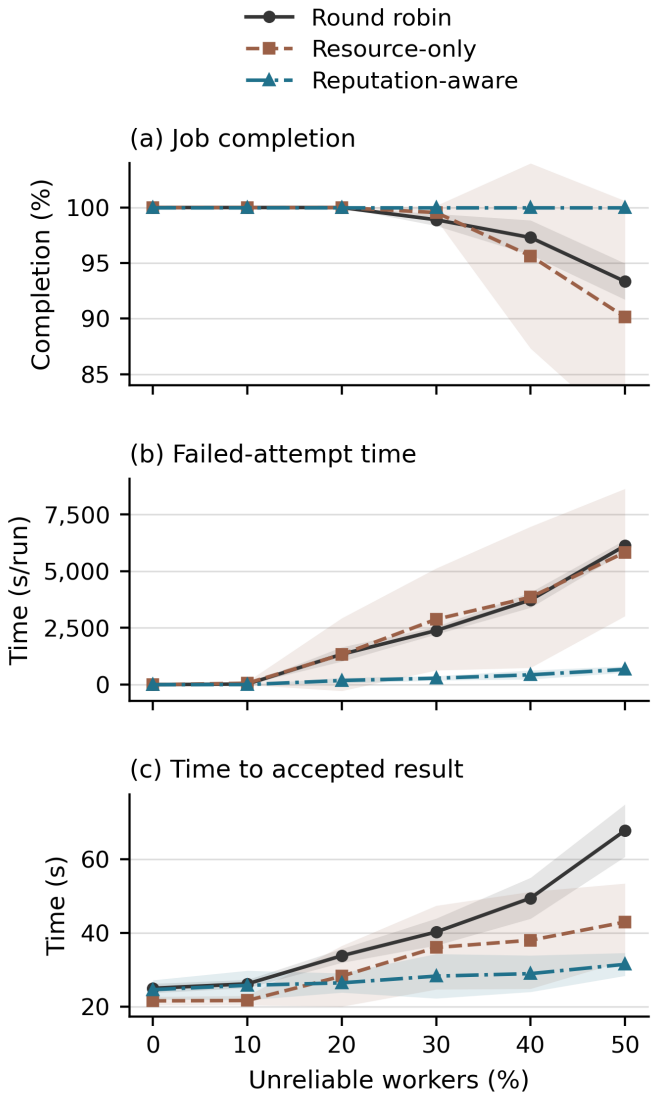

Figure 3. E2 scheduling performance across unreliable-worker fractions. Curves show means across ten seeds; shading indicates ±1 population standard deviation. Time to an accepted result includes successful jobs only.

5.3 E3 real fine-tuning and recorded lifecycle

The integrated real workload completed 120 jobs in 127 attempts. Honest workers supplied 120 accepted adapters. The remaining attempts comprised three rejected submissions, three timeouts, and one dropout outcome. All five programmed fault families were exercised at least once. Figure 4 separates their observed attempt counts; the experiment is a workflow demonstration, not a balanced detection benchmark for each failure family.

Accepted quality averaged 0.9998968, with a minimum of 0.9876159. These values indicate that the small held-out loss target was almost always reached, not that the model achieved nearly perfect predictive performance. The score's upper clipping makes it inappropriate for distinguishing improvements beyond the full-credit target.

Mean recorded time to an accepted result was 1.5132 seconds, with a median of 1.3921 seconds. Total control-plane time was 38.7951 seconds and recorded computation time was 135.0966 seconds, giving an overhead ratio of 28.72%. Averaging by submitted jobs gives approximately 0.3233 seconds of control-plane cost per job. Short toy jobs make fixed validation and transaction costs visible rather than negligible.

Training consumes actual GPU time. Fault timing, however, is represented by calibrated durations and virtual deadlines, avoiding deliberate GPU idling while waiting for a simulated dropout. Workers are logical behaviors on one controlled host. The result supports integration of training, validation, retries, and EVM recording, but does not estimate network latency, multi-organization throughput, or independent provider isolation.

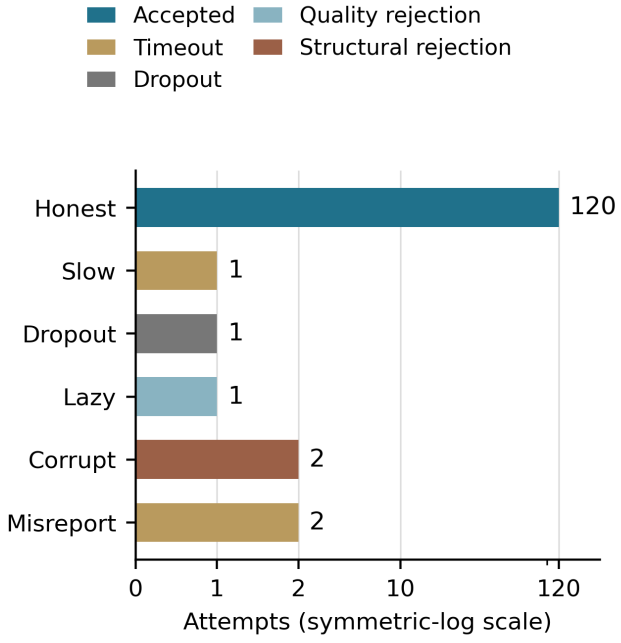

Figure 4. E3 attempt outcomes by worker class (127 attempts across 120 jobs). Counts are shown on a symmetric-log axis.

5.4 E4 ledger cost and amortization

The seven local EVM operations had median latencies between 18.63 and 19.70 milliseconds. Summing

the medians of the six operations for a normal job, excluding worker registration, gives 115.08 milliseconds. The corresponding sum of operation median gas values is 374,853. Deployment, first-time registration, and extra attempts are excluded from this normal-path estimate. Figure 5 gives the operation-level measurements.

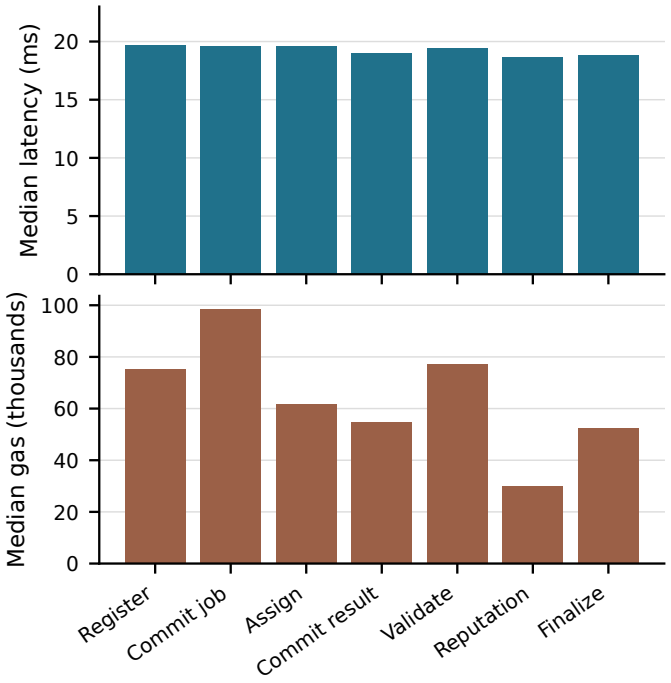

Figure 5. E4 local EVM operation costs, with 200 repetitions per operation. The normal six-operation job excludes registration. Latency medians are local execution measurements, and gas is not a currency price.

The 115.08-millisecond value is a sum of operation medians, not a directly observed percentile of end-to-end job latency. For a reference computation lasting 24 seconds, it corresponds to approximately 0.48% ledger-only overhead. By comparison, E3 includes model validation and other control work, which raises its average per-job control-plane cost to 0.3233 seconds.

If that E3 average were held fixed, keeping control-plane time below 5% of computation time would require computation longer than approximately 6.47 seconds. This is an amortization calculation, not a measurement of a long-job deployment. Validation cost may change with model size, evaluation-set size, caching, and device contention, while transaction latency may change with the ledger environment.

The py-evm setup has no external node communication, public mempool, or network confirmation delay. Its gas accounting and contract execution demonstrate the local implementation, but the measured latency is not a public-chain service-level estimate. A production evaluation would need explicit network and finality assumptions, and possibly transaction batching for small jobs.

5.5 E5 acceptance override

E5 crosses scheduling with an acceptance override. At nonzero unreliable fractions containing invalid submitters, invalid acceptance is zero with normal validation and one with the override. Table 4 shows the 50% condition. The state-based completion rate can improve when bad artifacts are counted as successful deliveries, demonstrating why completion alone is insufficient.

Table 4. E5 at 50% unreliable workers. Override forces acceptance and therefore changes the meaning of completion. Each arm contains ten runs of 200 jobs.

Policy	Validation	Completion (%)	Invalid accepted (%)
Round robin	Normal	93.35	0
Round robin	Override	99.35	100
Reputation-aware	Normal	100.00	0
Reputation-aware	Override	100.00	100

Reputation cannot correct a verdict that the platform has deliberately forced to success. If invalid work generates a positive outcome, history ceases to represent actual accepted quality. This makes the validator a foundation for the feedback loop rather than a replaceable reporting step.

The override executes some structural-check work before replacing the verdict. We therefore use E5 to study the removal of rejection semantics, not the speed benefit of bypassing validation entirely. Reduced failed attempt time in an override arm can likewise result from classifying bad work as accepted, rather than from avoiding wasted physical computation.

5.6 E6 adaptive behaviors and exploration

Identity resets expose an interaction between uncertainty and eligibility. A newly reset provider again has little evidence, so an optimism bonus can dominate the score. At 30% whitewashing workers in the undiscounted main grid, the pooled fraction of attempts assigned to whitewashers was 4.50% for c = 0.10, 90.18% for c = 0.25, and 96.69% for c = 0.50. Figure 6 presents the corresponding run-level means and spread, which use different weighting from those pooled proportions.

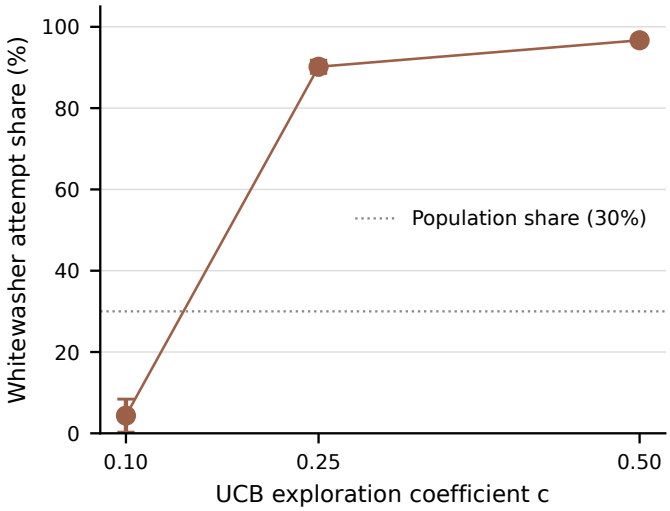

Figure 6. UCB sensitivity with 30% whitewashing workers and no evidence discount. Points are mean attempt shares across ten runs; bars show ±1 population standard deviation. The dotted line is population share, not a theoretical decision threshold.

These measurements show strong sensitivity in this particular setting. They do not prove that a universal threshold exists between 0.10 and 0.25. The UCB index is combined with normalized queue-dependent features, and workers change eligibility as jobs arrive and finish. An analytic cutoff that ignores these dynamics is not established by three sampled coefficients.

At 50% whitewashing workers, the pooled completion rate was 97.10% for scalar reputation with ϵ-greedy exploration and 97.95% for beta reputation with Thompson sampling. UCB completion was 84.70%, 10.40%, and 4.40% for c = 0.10, 0.25, and 0.50, respectively. Persistent laziness produced different behavior: UCB with c = 0.10 completed 99.85% of jobs at the same unreliable fraction. The failure is thus tied to the interaction with resetting history, rather than an assertion that exploration or UCB is always harmful.

Conditional defection experiments require additional care because a worker in the adversarial class can submit valid work on nondefecting attempts. Its assignment share must not be labeled an invalid result acceptance rate. In conditional defection scenarios, submission validity is classified using attempt-level behavior rather than worker identity alone. Results are reported separately by scenario and configuration. The 440 case validation benchmark refers exclusively to E1.

5.7 E7 payment rules and accepted quality

Under the main cost model, the shirker's chosen effort is 0.40 for flat payment and 0.70 for proportional payment. Table 5 separates the quality of accepted shirker outputs from quality across the entire platform. At 40% shirkers, the former rises from 0.5521 to 0.9991 while the latter rises from 0.8211 to 0.9996. Confusing these denominators would overstate the platform-wide effect of changing the fee.

Table 5. E7 shirker scenario at computing cost 0.3, zero penalty, and zero registration cost. Values are means across ten runs. Quality is normalized and capped at one.

Shirker share	Payment	Shirker Q	Overall Q
20%	Flat	0.5526	0.8900
20%	Proportional	0.9988	0.9998
40%	Flat	0.5521	0.8211
40%	Proportional	0.9991	0.9996

Increasing the failure penalty from zero to one did not change the selected effort or these quality outcomes in the pure shirker setting. A passing submission does not incur the failure penalty. Flat payment therefore continues to reward the least-cost accepted grid choice, while proportional payment rewards additional quality until its modeled marginal value no longer justifies extra effort. Figure 7 visualizes the resulting quality difference.

The production table is part of the behavior model, and the selected effort is a programmed best response to that table. This is evidence about the specified mechanism and its interaction with the simulator, not a behavioral experiment with human providers. E7 uses a fixed lookup table derived from synthetic backend calibration rather than measurements of real LoRA training. Under the main setting of $w = 1$, $k = 0.3$, $\tau = 0.5$, and zero failure penalty, the implemented lookup table yields unique optimal grid efforts of 0.40 for flat payment and 0.70 for proportional payment. The reported E7 results use this table. The effort–quality relationship remains a simulation assumption and has not been validated using real LoRA training.

Failure penalties and registration costs reduce the accounted profit of programmed invalid submitters. They do not directly change routing, and those actors do not stop participating after a loss. No empirical deterrence or market-wide welfare conclusion follows from accounting alone. A refundable stake, a permanent registration fee, and an opportunity cost of locked funds would also have different economic effects; only the implemented permanent charge is

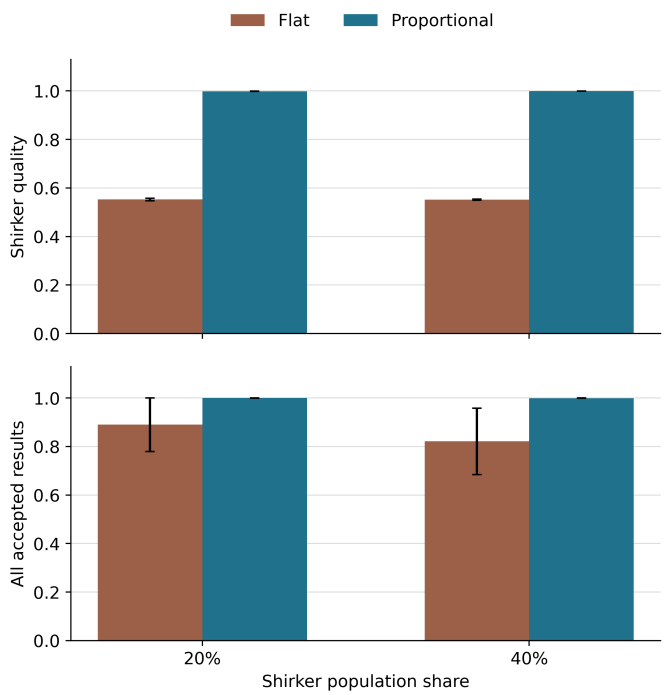

Figure 7. Payment effects in E7 with computing cost 0.3, zero failure penalty, and zero registration cost. Top: accepted shirker quality. Bottom: quality of all accepted submissions. Bars are run means and error bars are ±1 population standard deviation over ten seeds.

represented here.

6 Discussion

6.1 What the integrated design establishes

The experiments support distinct roles for validation, scheduling, and incentives. E1 and E5 concern whether a submitted adapter should be accepted. E2 concerns how much failed work occurs before acceptance. E7 concerns the quality delivered within the accepted region. A system can perform well on one dimension and poorly on another: E6 illustrates that rejecting bad results still leaves room for repeated assignment costs.

A practical consequence is to report more than final success. An operator should distinguish accepted artifact quality, attempts per job, unresolved jobs, and platform-accounted waiting. A payment rule should specify whether it purchases threshold compliance or rewards quality above that threshold. A reputation update should also be tied to a meaningful validation outcome, because a high score produced from uninformative success labels can reinforce bad selection.

6.2 Security and privacy boundaries

An adapter may satisfy the compatibility and held-out quality requirements while still containing a backdoor or performing poorly on another distribution. It may also have been obtained through a different training procedure. The present validator does not test these properties. Finite fault injection evaluates the constructed categories, not an unrestricted adaptive adversary targeting the full model behavior.

The dataset is public, the coordinator and validator are trusted, and the EVM owner supplies verdicts. Hash commitments do not provide confidentiality, identity authenticity, or proof of correct training. Real deployment would require authentication, access controls, artifact isolation, and an explicit evaluation-data policy. Those mechanisms should be assessed under a revised threat model rather than implicitly credited to the current prototype.

Whitewashing resets history within a fixed modeled population. General Sybil resistance [17] requires assumptions about who can create identities and how their influence is constrained. Entry costs may impose losses but can also burden honest newcomers, as the cheap-pseudonym literature emphasizes [16]. The current accounting study does not resolve that admission tradeoff.

6.3 Workload and statistical validity

Large grids use the synthetic backend. Their absolute effects depend on lazy submission speed, slow-worker multipliers, job arrivals, queue limits, and the shape of the effort-quality relation. Real training uses one model, one corpus configuration, short runs, and logical workers on one host. Neither setup measures heterogeneous production traffic across independent organizations.

The all-honest latency result and the different non-reputation weights limit claims about policy dominance. The hardware term is effectively constant after eligibility screening. A controlled weight-matched ablation, arrival-rate sweep, and larger workload set would help establish where reputation contributes most. Each configuration is evaluated over ten seeded runs. Tasks within a run share a reputation history and therefore cannot be treated as independent experiments.

6.4 Engineering and economic scope

The simulator has open-loop arrivals and deadline-based dropout detection. Backpressure,

heartbeats, progress evidence, and cooperative cancellation may change both waiting and failure classification. Local EVM timing excludes external communication and finality. Verification cost can also scale with model size, so a constant overhead extrapolation should not be treated as a deployment guarantee.

Economic results are conditional on a tabulated production model and linear effort cost. No real payments occur. Invalid submitters retain their programmed strategy even when unprofitable, and the entry-cost implementation has no refund. The work demonstrates a difference between flat and proportional rewards in this model, but does not establish truthful bidding, budget feasibility, participation equilibrium, or welfare optimality.

7 Conclusion

TrustCompute integrates adapter validation, reputation-aware scheduling, bounded retries, and auditable task records for outsourced LoRA fine-tuning. In the finite validation benchmark, its structural and quality stages separated all constructed invalid cases from valid cases. In simulations with unreliable workers, the reputation-aware configuration reduced failed attempt time and retries while maintaining task completion. The real fine-tuning workload demonstrated the integrated lifecycle with local EVM recording.

The extensions identify two limits that matter for platform design. Cheap identity resets can turn strong exploration into repeated exposure to unreliable providers, even when their outputs are rejected. Flat payment can reward minimally acceptable effort, whereas a quality proportional rule changes the best response under the specified cost model. These findings support evaluating acceptance, assignment cost, and delivered quality together, with explicit denominators and trust assumptions.

Future work should prioritize matched weight scheduling comparisons, calibrated real training effort curves, and multi node execution with progress reporting. Stronger process guarantees would require mechanisms beyond final artifact validation. The present evidence establishes a controlled platform prototype and its observed tradeoffs, not a proof of training or a deployment wide security guarantee.

Data Availability Statement

The data used to support the findings of this study are available from the corresponding author upon request.

Funding

This work was supported without any funding.

Conflicts of Interest

Kuo-Hui Yeh served as an Editor-in-Chief of the *Journal of Reliable and Secure Computing* at the time of manuscript submission. To ensure the integrity of the peer-review process, Kuo-Hui Yeh was not involved in the editorial handling, peer review, or decision-making process for this manuscript, which was handled independently by another editor. The remaining authors declare no conflicts of interest.

AI Use Statement

The authors declare that Claude Opus 5, developed by Anthropic, was used for drafting and translation during the preparation of the manuscript. The authors have carefully reviewed, revised, and verified the AI-assisted output and take full responsibility for the content of the manuscript.

Ethical Approval and Consent to Participate

Not applicable.

References

[1] Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2021). Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*. [CrossRef]

[2] Kang, J., Xiong, Z., Niyato, D., Xie, S., & Zhang, J. (2019). Incentive mechanism for reliable federated learning: A joint optimization approach to combining reputation and contract theory. *IEEE Internet of Things Journal, 6*(6), 10700-10714. [CrossRef]

[3] Anderson, D. P. (2004, November). Boinc: A system for public-resource computing and storage. In *Fifth IEEE/ACM international workshop on grid computing* (pp. 4-10). IEEE. [CrossRef]

[4] Borzunov, A., Baranchuk, D., Dettmers, T., Riabinin, M., Belkada, Y., Chumachenko, A., ... & Raffel, C. (2023, July). Petals: Collaborative inference and fine-tuning of large models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)* (pp. 558-568). [CrossRef]

[5] Ryabinin, M., & Gusev, A. (2020). Towards crowdsourced training of large neural networks

using decentralized mixture-of-experts. *Advances in Neural Information Processing Systems, 33*, 3659-3672.

[6] Weng, J., Weng, J., Zhang, J., Li, M., Zhang, Y., & Luo, W. (2019). Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive. *IEEE Transactions on Dependable and Secure Computing, 18*(5), 2438-2455. [CrossRef]

[7] Parno, B., Howell, J., Gentry, C., & Raykova, M. (2016). Pinocchio: Nearly practical verifiable computation. *Communications of the ACM, 59*(2), 103-112. [CrossRef]

[8] Tramer, F., & Boneh, D. (2018). Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287*. [CrossRef]

[9] Zhao, L., Wang, Q., Wang, C., Li, Q., Shen, C., & Feng, B. (2021). Veriml: Enabling integrity assurances and fair payments for machine learning as a service. *IEEE Transactions on Parallel and Distributed Systems, 32*(10), 2524-2540. [CrossRef]

[10] Jia, H., Yaghini, M., Choquette-Choo, C. A., Dullerud, N., Thudi, A., Chandrasekaran, V., & Papernot, N. (2021, May). Proof-of-learning: Definitions and practice. In *2021 IEEE Symposium on Security and Privacy (SP)* (pp. 1039-1056). IEEE. [CrossRef]

[11] Körbel, B., Sigwart, M., Frauenthaler, P., Sober, M., & Schulte, S. (2021, November). Blockchain-based result verification for computation offloading. In *International Conference on Service-Oriented Computing (pp. 99-115)*. Cham: Springer International Publishing. [CrossRef]

[12] Kamvar, S. D., Schlosser, M. T., & Garcia-Molina, H. (2003). The EigenTrust algorithm for reputation management in P2P networks. *Proceedings of the 12th International Conference on World Wide Web*, 640–651. [CrossRef]

[13] Josang, A., & Ismail, R. (2002, June). The beta reputation system. In *Proceedings of the 15th bled electronic commerce conference* (Vol. 5, pp. 324-337). https://domino.fov.um.si/proceedings.nsf/Proceedings/D9E48B66F32A7DFFC1256E9F00355B37/$File/josang.pdf

[14] Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning, 47*(2), 235-256. [CrossRef]

[15] Daniel, J. R., Benjamin, V. R., Abbas, K., Ian, O., & Zheng, W. (2018). A tutorial on thompson sampling. *Foundations and trends® in machine learning, 11*(1), 1-99. [CrossRef]

[16] Friedman, E. J., & Resnick, P. (2001). The social cost of cheap pseudonyms. *Journal of Economics & Management Strategy, 10*(2), 173-199. [CrossRef]

[17] Douceur, J. R. (2002, March). The sybil attack. In *International workshop on peer-to-peer systems* (pp. 251-260). Berlin, Heidelberg: Springer Berlin Heidelberg. [CrossRef]

[18] Ding, N., Fang, Z., & Huang, J. (2020). Optimal contract design for efficient federated learning with multi-dimensional private information. *IEEE Journal on Selected Areas in Communications, 39*(1), 186-200. [CrossRef]

[19] Qwen Team. (2024). Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*. [CrossRef]

[20] Merity, S., Xiong, C., Bradbury, J., & Socher, R. (2016). Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*. [CrossRef]

Lin-Fa Lee received the B.S. and M.S. degrees from National Dong Hwa University, Hualien, Taiwan. He is currently pursuing the Ph.D. degree with the Institute of Artificial Intelligence Innovation, National Yang Ming Chiao Tung University, Hsinchu, Taiwan. His research interests include AI security, privacy-preserving machine learning, and blockchain systems. (Email: prologue.ii14@nycu.edu.tw)

Yi-Yu Chang received the B.S. degree from National Dong Hwa University, Hualien, Taiwan, and the M.S. degree in computer science from National Chengchi University, Taipei, Taiwan. He is currently pursuing the Ph.D. degree with the Institute of Artificial Intelligence Innovation, National Yang Ming Chiao Tung University, Hsinchu, Taiwan. His research interests include machine learning, natural language processing (NLP), and blockchain. (Email: daniel282907@gmail.com)

Kuo-Hui Yeh received the M.S. and Ph.D. degrees in information management from the National Taiwan University of Science and Technology, Taipei, Taiwan, in 2005 and 2010, respectively. He is currently a Professor with the Institute of Artificial Intelligence Innovation, National Yang Ming Chiao Tung University, Hsinchu, Taiwan. Prior to this appointment, he was a Professor with the Department of Information Management, National Dong Hwa University, Hualien, Taiwan, from February 2012 to January 2024. He has contributed over 150 articles to esteemed journals and conferences, covering a wide array of research interests such as the IoT security, blockchain, NFC/RFID security, authentication, digital signatures, data privacy, and network security. Furthermore, he plays a pivotal role in the academic community, serving as an Associate Editor (or Editorial Board Member) for several journals, including the Journal of Information Security and Applications, Human-Centric Computing and Information Sciences, Symmetry, Journal of Internet Technology, and CMC-Computers, Materials and Continua. In the professional realm, he holds memberships with (ISC)2, ISA, ISACA, CAA, and CCISA. His professional qualifications include certifications, such as CISSP, CISM, Security+, ISO 27001/27701/42001 Lead Auditor, IEC 62443-2-1 Lead Auditor, and ISA/IEC 62443 cybersecurity expert, covering fundamentals, risk assessment, design, and maintenance specialties. (Email: khyeh@nycu.edu.tw)