

DisTrust: Adaptive Trust-Based Verification for Resilient Distributed Computation

Fouad Afiouni
Purdue University
West Lafayette, IN, USA
fafiouni@purdue.edu

Saurabh Bagchi
Purdue University
West Lafayette, IN, USA
sbagchi@purdue.edu

Aravind Machiry
Purdue University
West Lafayette, IN, USA
amachiry@purdue.edu

Abstract—Autonomous systems that offload computation to distributed edge, cloud, or volunteer nodes face a fundamental runtime resilience problem: verifying results from untrusted workers is costly and unreliable under Byzantine behavior, especially when adversaries adapt to evade detection. Existing approaches such as fixed replication incur constant overhead, while simpler auditing schemes fail to eliminate strategic adversaries. We propose DisTrust, a self-adaptive trust-based verification mechanism that dynamically redistributes work based on agent reliability, increasing scrutiny on suspicious nodes and progressively reducing their influence until they are ejected. The system requires no prior knowledge of the number or behavior of adversaries and naturally adjusts its verification effort to the observed threat level. We show that DisTrust suppresses and ejects gaming and omniscient Byzantine adversaries below the 50% threshold, while reducing verification overhead and avoiding the persistent damage and inefficiency of fixed replication baselines.

Index Terms—Autonomous systems, Byzantine fault tolerance, Trust management, Adaptive verification, Resilient coordination

I. INTRODUCTION

Autonomous system deployments where resource-constrained agents offload heavy computation to external nodes face a fundamental verification problem. A delivery drone fleet may offload path planning to third-party edge servers; a robotic warehouse may delegate perception tasks to nearby compute nodes; a sensor network may farm out data processing to a volunteer GPU pool. In each case, some fraction of compute nodes may be continually faulty, intermittently unreliable, or deliberately compromised. Trust and reputation mechanisms have been proposed for such multi-agent settings [8]–[10], including centralized schemes that eject low-reputation nodes [13], [14]. However, these approaches typically treat trust as a cumulative score and do not couple trust directly to computational influence. A low-reputation node may still receive full workloads until a separate decision layer acts on the score. Fixed replication tolerates faults but never removes the faulty workers, who remain in the system indefinitely consuming resources.

This problem is well studied in volunteer and outsourced computing more broadly. Platforms such as SETI@home and BOINC [1], [2] address it through fixed-rate replication, assigning each task to k workers and accepting the majority result. Cryptographic approaches such as verifiable computa-

tion [3] provide formal correctness proofs but require circuit-level encodings. Coding-theoretic methods such as Draco [4] and Detox [5] recover correct results under bounded failures but are restricted to specific algebraic structures.

We propose *DisTrust*, a lightweight, problem-agnostic trust management layer for systems where a coordinator dispatches computational tasks to autonomous agents. The core idea is to model trust as a scarce currency managed by the coordinator. Agents begin with an initial trust endowment and compete for trust throughout the system's lifetime. Agents with lower trust receive more verification (check) work and less real work, gradually reducing their influence on the system's outputs. When an agent's results disagree with the majority, it loses trust. When trust is fully depleted, the agent is ejected. DisTrust accumulates information from past interactions, but differs from traditional reputation systems in how that information is used: trust is modeled as a scarce economic resource with built-in scarcity pressure: trust decays over time, penalties exceed rewards ($loss > reward$), and trust directly determines an agent's share of real work. This creates an asymmetric game in which any sustained nonzero attack rate leads to net trust loss over time. While honest agents may occasionally lose trust through unfavorable committee assignments, they recover through successful checks; adversarial agents face a structural deficit that honest behavior cannot offset without ceasing the attack. The mechanism is not observational; it is interventional and *self-adaptive*: low trust does not merely flag an agent for review, it actively reduces the agent's influence on the computation. This feedback loop – lower trust leads to more verification, which leads to faster detection, which leads to further trust loss – accelerates against adversaries and quiets when the system is clean. The coordinator can also tune the aggressiveness of the defense through a small set of interpretable hyperparameters, allowing operators to balance verification overhead against detection speed based on mission requirements.

We evaluate DisTrust against two standard baselines. The first is *fixed k -replication*, where every task is assigned to k agents and the majority-vote result is accepted. This is the most widely used verification strategy in coordinator-based distributed systems [1], [2]. Fixed replication masks faulty results but never identifies or removes the faulty agents, which remain in the system indefinitely, consuming resources and

contributing damage whenever they form a local majority. The second is *threshold-based auditing*, where agents are checked with a fixed probability and ejected after accumulating a threshold number of detected mismatches (strikes), with optional strike cooldown. This represents the natural “engineering baseline” that a practitioner might implement. For workload-agnostic, coordinator-based distributed computation, fixed replication and threshold auditing are the most relevant practical baselines; other approaches discussed in Section II either solve a different problem or require stronger structural assumptions. We equip the Byzantine adversary with a *gaming strategy*: it monitors its own strike count and switches to correct behavior when approaching the ejection threshold. Our results show that DisTrust successfully ejects gaming adversaries, while fixed replication accumulates up to $62 \times$ more corrupted results accepted into the final computation and threshold-based auditing produces false positives while failing to eject any adversary. When layered on top of existing robust aggregation defenses, DisTrust drives every method to clean-system output quality — a result no standalone defense achieves.

We make the following contributions:

- A trust-as-currency framework for adaptive verification, designed as a pluggable layer compatible with any coordinator-based task dispatching architecture.
- An m -majority verification scheme that provides robust fault attribution even when adversarial agents collude.
- A comparative evaluation against fixed replication and threshold-based auditing, including gaming and omniscient Byzantine adversaries.
- A drift-based analysis explaining why honest agents retain trust and adversarial agents are ejected below the $N/2$ boundary.

II. RELATED WORK

A. Verifiable Outsourced Computation

The problem of verifying results returned by untrusted workers was formalized by Gennaro et al. [3] under the name *verifiable computation*, but these schemes require circuit-level representations and impose non-trivial overhead. In practice, distributed computing platforms such as SETI@home and BOINC [1], [2] have relied on replication-based verification, where each task is assigned to multiple workers and results are compared through majority voting. This approach is simple and effective but imposes fixed overhead regardless of threat level. Coding-theoretic approaches such as Draco [4] and Detox [5] recover correct results under bounded failures, but are restricted to computations with specific algebraic structures. Our work takes a different approach: rather than correcting faulty results or proving correctness, we *detect and eject* faulty agents over time using adaptive redundancy that self-tunes based on accumulated trust.

B. Byzantine Fault Tolerance in Distributed Systems

Byzantine fault tolerance (BFT) protocols guarantee agreement under bounded adversarial participation [6], [7], typically

tolerating $f < n/3$ Byzantine nodes. In autonomous systems, BFT has been applied to multi-robot coordination [8]. However, these approaches focus on *tolerating* faults rather than *removing* faulty agents. Our approach complements BFT by ejecting compromised agents over time, reducing the adversarial population.

C. Trust and Reputation in Multi-Agent Systems

Trust and reputation mechanisms are widely used in multi-agent and multi-robot systems to estimate the reliability of participants [9], [10]. Agents typically rate peers based on past interactions, and accumulated reputation scores guide future task assignments or coalition formation. In autonomous vehicle platoons and drone swarms, trust models have been proposed to detect misbehaving members and exclude them from cooperative tasks [11].

DisTrust draws on the idea of trust as a system resource but departs from collaborative reputation models in two key ways. First, trust is managed centrally by the coordinator rather than through peer-to-peer agreement, which avoids the overhead and vulnerability of distributed reputation protocols. Second, trust is modeled as a *scarce, decaying currency* rather than an accumulated score: agents must continuously re-earn trust through successful verifications, and trust naturally decays over time to ensure ongoing accountability. This makes the system resistant to adversaries who build trust during a dormant phase before launching an attack. Slashing mechanisms in blockchain protocols [16] serve a similar function — staking participants lose deposited assets for misbehavior — though these operate in consensus settings with cryptographic accountability rather than workload-agnostic task dispatch.

III. PROBLEM SETTING

A. System Setting

We begin by describing our network architecture and the possible node profile. We deal with a server-based system where a central coordinator distributes tasks to autonomous worker nodes (Fig. 1). Nodes can be *correct*, *continually faulty* (always return incorrect results), *randomly faulty* (fail intermittently, e.g., due to hardware errors), or *Byzantine* (strategically return incorrect results, potentially under coordinated adversarial control). DisTrust can operate standalone or as an augmentation layer on top of existing defenses. Systems that already tolerate some faults gain accelerated adversary removal; systems with no existing defense gain runtime resilience against partial compromise.

B. Threat Model

Continually faulty and randomly faulty agents are special cases evaluated in Section VII to demonstrate DisTrust’s behavior across the fault severity spectrum. The threat model below focuses on the hardest case: colluding Byzantine agents. We consider an adversary that controls strictly fewer than $N/2$ worker nodes in the system. Compromised nodes may return arbitrary, incorrect results for any task, and may collude to produce consistent but incorrect outputs. We consider an

omniscient adversary that has full knowledge of the system state at any time, including all trust values, task assignments, and communication contents. However, the adversary is not omnipotent: it cannot directly modify the coordinator's state, impersonate other nodes, or interfere with communications between honest nodes and the coordinator. We assume that non-omniscient agents cannot distinguish between different types of tasks dispatched by the coordinator; this holds when the coordinator controls task formatting and routing. The omniscient adversary relaxes this assumption, as evaluated in Section VII. The check-work mechanism requires deterministic computation: correct agents given the same input must produce identical outputs. Non-deterministic workloads would require approximate matching, which we leave to future work.

C. Guarantees

Since our solution is problem-agnostic, it does not *guarantee* the correctness of the aggregate computation or that no corrupted results were accepted. This is consistent with the majority of prior work in this space: replication approaches accept majority-vote results without correctness guarantees, and trust-based approaches such as [8], [11] adjust reputation scores without guaranteeing output correctness. Methods that do provide guarantees are restricted to specialized settings: Draco [4] and Detox [5] guarantee recovery only for computations expressible as specific algebraic structures, and cryptographic verifiable computation [3] requires circuit-level encodings impractical for most deployments. No existing general-purpose, workload-agnostic defense provides formal output correctness guarantees.

IV. TRUSTCOIN

The central intuition behind DisTrust is to model trust as a scarce currency in the system, managed exclusively by the coordinator. All agents compete for trust throughout the system's lifetime in order to continue receiving work. In the remainder of the paper, we use coins, trust coins, and trust interchangeably. An agent earns trust coins by successfully checking the work of another agent, and loses trust when it is determined, with some confidence, to be defective or malicious. The exact scheme and quantification by which trust is gained or lost is maintained on the coordinator and is entirely hidden from the agents. The rules governing trust dynamics are as follows:

- Each agent has a configurable upper bound on trust coins. Trustworthiness is the ratio of current coins to this maximum, and directly determines the agent's share of real work. By assigning different maximums, operators encode prior trust: e.g., a managed cloud instance with a maximum of 10 coins reaches full trust more easily than a volunteer device with a maximum of 30, biasing workload toward more trusted platforms. Unless stated otherwise, *trust* refers to this ratio.
- At initialization ($t = 0$), coins are distributed evenly among all available agents. Based on its current trust percentage, each agent is assigned either *real work*,

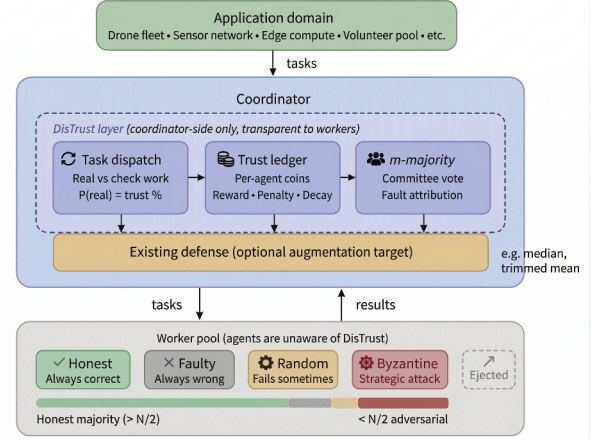


Fig. 1. DisTrust system architecture. The trust mechanism resides entirely on the coordinator and is transparent to workers. An optional existing defense (e.g., coordinate-wise median) can be augmented by DisTrust. The bottom bar illustrates one possible population composition; DisTrust operates under any mix where adversarial agents comprise fewer than $N/2$.

consisting of new tasks whose results contribute to the final output, or *check work*, consisting of tasks previously completed by another agent and stored by the coordinator. The probability of receiving real work equals the agent's trust percentage. As an agent's trust decreases, it receives proportionally less real work and more check work, progressively reducing its influence on the computation. Since agents cannot distinguish between real and check work, they remain oblivious to their trust level and the type of task they are performing.

- Suppose agent A completes a check task that was previously performed by agent B and returns its result to the coordinator. If the results match, agent A is rewarded with r coins from the coordinator's reserve. If the coordinator's reserve is empty, the reward is skipped. If the results do not match, one of the two agents must be continually faulty or Byzantine. In the base case, agent A is assumed truthful: agent A receives no reward, and agent B loses l coins, which are transferred to the coordinator. We set $l > r$ so that trust is lost faster than it is gained, accelerating the ejection of suspected agents. In the general case, we employ an m -majority scheme (described below) in which m agents vote to determine which party is at fault before any coin transfer occurs.
- Trust decays naturally over time. After every δ completed tasks, each agent loses λ coins, which are transferred to the coordinator's reserve:

$$T_{\text{agent}_i} = T_{\text{agent}_i} - \lambda \quad \text{every } \delta \text{ tasks by agent}_i \quad (1a)$$

This ensures that all agents are periodically re-verified, preventing any agent from accumulating trust indefinitely without ongoing validation.

Under these rules, when all agents are correct, the system reaches a stable equilibrium in which the ratio of real work

to check work remains approximately constant. When a malicious agent returns incorrect results, it progressively loses trust, receives an increasing proportion of check work (which does not affect the final output), and is eventually ejected from the system entirely. Importantly, this attenuation is proportional: an intermittently faulty agent that fails occasionally settles at a moderate trust level and continues to contribute at reduced capacity, while a consistently malicious agent is rapidly driven to zero trust and ejected. The mechanism does not require the operator to specify fault types, the trust dynamics naturally distinguish between them. The parameters l , r , δ , and λ control the aggressiveness of this process and determine how quickly a faulty agent is detected and removed. Poorly chosen parameters may lead to false ejections of correct agents in adversarial conditions. Three properties distinguish this design from prior work: (1) trust is *operative* — it directly controls workload share, not just an advisory score; (2) economics are *asymmetric* ($l > r$), so alternating good and bad behavior still yields net trust loss [15]; and (3) continuous decay prevents coasting on historical trust.

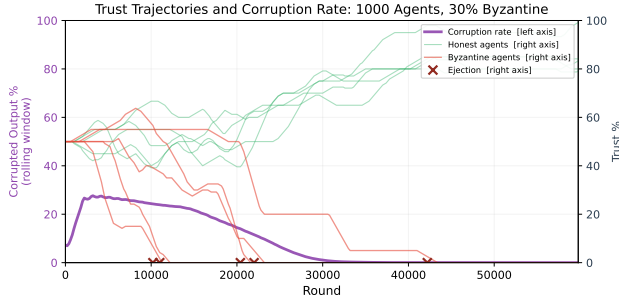


Fig. 2. Trust trajectories and corruption rate under 30% Byzantine agents. Trust of adversarial agents (red) declines to ejection while honest agents (green) remain stable. The cumulative corrupted output fraction (purple) drops as adversaries are suppressed and ejected.

m-majority To improve the robustness of verification checks, we require that a match or mismatch be settled only when a majority of m agents agree on a single value, i.e., at least $\lceil m/2 \rceil + 1$ agents. If no majority is established (for instance, if all m agents return distinct results), additional agents are recruited until a majority forms. The majority threshold remains fixed at $\lceil m/2 \rceil + 1$ and is not increased as more agents join the check. The rationale is that if all m checking agents disagree with each other, this does not constitute meaningful evidence against the original agent. Since the computations are deterministic, correct agents must produce identical results.

To accelerate the removal of faulty agents, any agent that participated in a check and did not agree with the majority loses l coins, regardless of whether the majority result matched or mismatched the original. The reasoning is that an agent whose result disagrees with the majority would also fail if it were itself the subject of a check, so penalizing it immediately is warranted.

V. DESIGN AND SCALABILITY

The dispatch logic, which constitutes the core of our resilience mechanism, is independent of any computation performed by the agents. The coordinator can therefore pre-fetch and prepare task assignments in parallel with ongoing agent computation, allowing the following operations to be fully pipelined.

The coordinator maintains per-agent trust values, decay counters, and a fixed-size stack of recent results. All logic resides on the coordinator side; workers are unmodified. When the coordinator’s coin reserve is empty, agents simply receive real work, and the reserve replenishes through decay events. Randomness is introduced in task selection and scheduling to prevent the system from converging to a deterministic steady-state pattern.

Since all added logic resides exclusively on the coordinator side (specifically in the dispatch unit), scalability depends only on coordinator complexity. The coordinator requires constant space per agent. We summarize the time complexity of coordinator operations per cycle below.

n = Total number of agents in the system.

m = The m in m -Majority. E.g., $m = 3$ in 3-majority.

Operation	Avg.	Worst
(a) Check selection	$O(m)$	$O(n+m)$
(b) Majority & coins	$O(m)$	$O(n)$
(c) Full verification cycle	$O(m^2)$	$O(n^2)$

The average case follows from m iterations of subtasks (a) and (b); the worst case arises when up to n agents must be sampled. For 1-majority, the worst case for (b) reduces to $O(1)$.

VI. EXPERIMENTAL SETUP

We evaluate DisTrust through a prototype implementation in which the coordinator and each worker run as independent processes communicating over message queues, with workers performing real computation (local PCA subspace estimation) on each dispatched task. The mechanism is workload-agnostic; the PCA workload is used only to exercise real dispatch, execution, and verification paths. We use 1000 workers under varying initial trust levels, adversarial fractions, and fault models. All agents are homogeneous with maximum coin capacity set to 100. Unless otherwise stated, the parameters l , r , δ , and λ (defined in Section IV) are set to 10, 5, 20, and 5, respectively. We evaluate three fault models: *faulty* agents (always return incorrect results), *randomly faulty* agents (fail with probability $1/3$), and *Byzantine* agents (return a coordinated incorrect result). We additionally implement an *omniscient Byzantine* adversary that, as described in the threat model, has full knowledge of the system state including task types, and exploits this by returning correct results on check work while corrupting real work. Workers are implemented as independent Python processes using the `multiprocessing` library; all timing and overhead measurements are wall-clock. The measured overheads reflect real inter-process communication and coordinator logic, though a production deployment

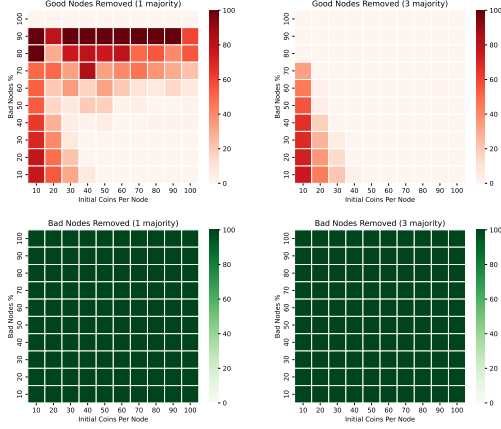


Fig. 3. Faulty (non-colluding) agents: ejection rates. Under 3-majority, all faulty agents are ejected with zero false positives at coins ≥ 30 .

with optimized networking would likely improve on these numbers.

VII. EVALUATION

A. Detection Under Different Fault Models

Recall that under m -majority, fault attribution requires agreement from at least $\lceil m/2 \rceil + 1$ agents (Section IV). Faulty and randomly faulty agents are the easiest cases. Under 3-majority, all consistently faulty agents are ejected with zero false positives at initial coins ≥ 30 , even at 90% adversarial fraction (Fig. 3). This is expected: non-colluding agents cannot form a majority and lose trust rapidly. Under 1-majority, false ejections of correct agents occur at low initial trust, where the aggressive verification rate causes honest agents to occasionally lose trust through unfavorable check assignments. Randomly faulty agents show similar behavior under 3-majority. Under 1-majority with high initial trust, some randomly faulty agents persist because the low verification rate allows intermittent failures to go undetected.

Fig. 2 shows individual trust trajectories for five honest and five Byzantine agents, along with the cumulative fraction of corrupted outputs accepted. Honest agents maintain stable trust, while Byzantine agents exhibit steady downward drift, eventually reaching zero and being ejected. The corrupted output fraction (purple) drops from approximately 30% to near zero as adversarial agents are progressively removed, illustrating the suppression effect: damage decreases well before the last adversary is ejected.

The Byzantine case (Fig. 4) is more interesting and exhibits a sharp transition at the 50% threshold, which we analyze in Section VIII. Below 50%, both 1-majority and 3-majority eject all Byzantine agents with minimal false positives at moderate to high initial trust. Above 50%, the adversarial majority ejects correct agents instead. The complementary structure of the upper (good removed) and lower (bad removed) heatmaps

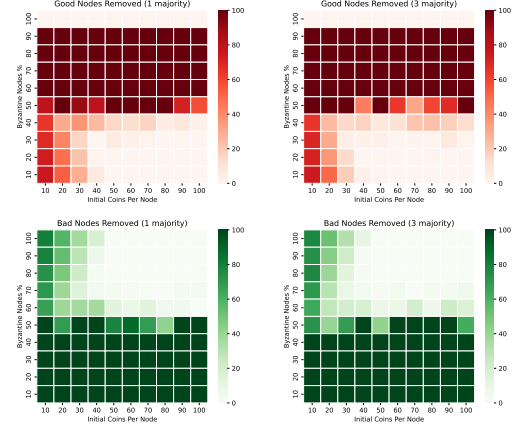


Fig. 4. Byzantine agents: ejection rates across varying adversarial fractions and initial trust levels, showing the $N/2$ phase transition.

illustrates this transition: the pattern is symmetric around the 50% boundary, consistent with the drift analysis.

B. Resilience Under Omniscient Adversary

To validate that omniscience does not help below $N/2$, we implement an optimal omniscient adversary that returns correct results on all check work. Ejection and false positive rates overlap almost perfectly with the non-omniscient case (Fig. 5), confirming that the adversary cannot prevent honest agents from verifying its corrupted real-work trail.

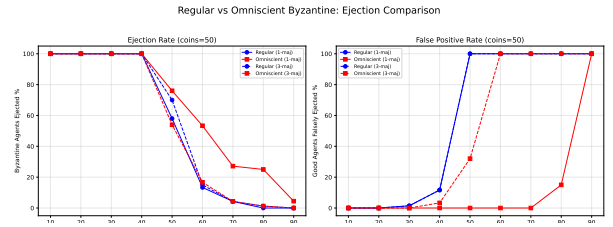


Fig. 5. Regular vs. omniscient Byzantine: ejection and false positive rates overlap below $N/2$, confirming that full system knowledge does not help the adversary.

C. Baseline Comparison

We compare DisTrust against two categories of baselines under a *gaming* Byzantine adversary that strategically alternates between correct and incorrect behavior to avoid detection.

Fixed replication (2x and 3x) assigns every task to k agents and accepts the majority-vote result. This is the standard approach used in distributed computing platforms. Fixed replication never ejects adversarial agents: they remain in the system indefinitely, consuming resources and contributing incorrect results whenever they form a local majority.

Threshold-based auditing checks agents with a fixed probability and tracks the number of detected mismatches (strikes)

per agent. An agent is ejected after accumulating a threshold number of strikes. We evaluate configurations with and without strike cooldown (periodic forgiveness). The gaming Byzantine adversary monitors its strike count and switches to correct behavior when approaching the ejection threshold. $\text{Thresh}(3)$ and $\text{Thresh}(5)$ refer to ejection thresholds of 3 and 5 strikes, respectively.

Fig. 6 shows the results at 20% and 40% Byzantine fractions. Among all evaluated methods, DisTrust is the only one that successfully detects and ejects gaming Byzantine agents. Fixed replication and threshold-based auditing both fail to eject any adversarial agents across all configurations. Results at 30% are consistent. All baseline results are averaged over 10 independent seeds; DisTrust achieves zero false positives across all seeds and configurations.

The consequences are significant: fixed 2x replication accepts 60,010 corrupted results at 30% Byzantine versus 963 for DisTrust, while imposing constant 50–67% overhead. Threshold auditing produces 2–18 false positives while failing to eject any adversary. DisTrust achieves the lowest false positive rate and highest effective throughput (696 real tasks per 1,000 dispatches at 30% Byzantine). As shown in the heatmaps, false positives can occur at very low initial trust (coins ≤ 20), where aggressive verification causes occasional misattribution.

DisTrust resists gaming because trust decays regardless of behavior, lower trust creates a verification feedback loop, and asymmetric penalties ($l > r$) ensure net trust loss for any nonzero attack rate. Unlike fixed replication and threshold auditing, where agents operate at full influence until removal, DisTrust continuously modulates each agent’s influence proportional to its trust.

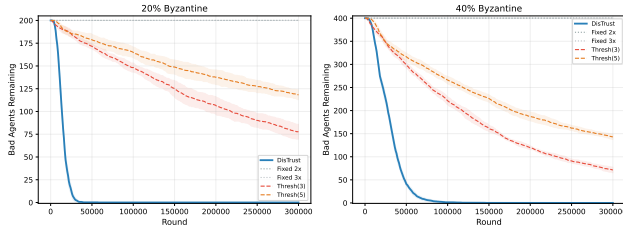


Fig. 6. Baseline comparison (averaged over 10 seeds with standard deviation) against a gaming Byzantine adversary at 20% and 40% adversarial fractions. DisTrust is the only evaluated method that ejects adversaries.

D. End-to-End Output Quality

We evaluate on a distributed sensor fusion task representative of autonomous system deployments: 1000 sensors collaboratively estimate a time-varying multi-dimensional signal, where each sensor computes a local estimate from its observations. This mirrors workloads in drone fleet telemetry aggregation, distributed environmental monitoring, and multi-robot state estimation. We compare two widely used robust aggregation rules — coordinate-wise median and trimmed mean [12] — which are standard defenses in Byzantine-resilient distributed estimation. We pair each defense with and without DisTrust to isolate the augmentation effect (Fig. 7).

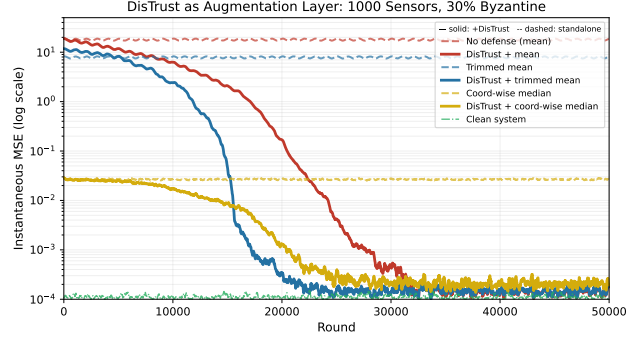


Fig. 7. End-to-end output quality on distributed sensor fusion (30% Byzantine). Dashed lines show standalone defenses; solid lines show the same defense augmented with DisTrust. Every augmented method converges to clean-system quality after adversary ejection.

Without any defense, MSE remains above 18 throughout. Coordinate-wise median [12], a widely used robust aggregator, reduces MSE to approximately 0.03 but plateaus there permanently because adversarial agents are never removed. When DisTrust is layered on top of the same defenses, every method converges to clean-system quality (< 0.001) after adversarial agents are ejected. DisTrust + coordinate-wise median achieves the best overall result: low initial MSE (the median handles early-phase corruption) and convergence to near-zero MSE once adversaries are removed. This demonstrates that DisTrust functions as a true augmentation layer: it improves the underlying defense by driving the system toward an attacker-free state with negligible coordinator-side cost.

E. Parameter Sensitivity

We sweep each of the four hyperparameters (l , r , δ , λ) individually while holding the others at their default values (Fig. 8). All sweeps use 30% Byzantine agents with 1-majority and moderate initial trust (coins=50), representing conservative operating conditions; higher initial trust or 3-majority further reduces false positives as shown in the heatmaps.

The penalty l is the dominant parameter and reveals a clear detection–accuracy tradeoff: increasing l from 2 to 30 reduces detection time from 60,000 rounds to 15,000 but increases false positives from 0 to 132 (19% of honest agents). At the default $l=10$, detection completes in 38,000 rounds with only 12 false positives (1.7% of honest agents). The reward r shows non-monotonic behavior: very low r yields fast detection because agents cannot accumulate trust and remain under heavy verification, while high r (≥ 8) prevents ejection entirely as Byzantine agents recover trust between detections. The decay interval δ has moderate impact, with shorter intervals accelerating detection. The decay amount λ has minimal effect across all values tested. Detection time ranges from 27,000 to 48,000 rounds across all δ values, and from 34,000 to 43,000 across all λ values. Importantly, the method is robust to suboptimal hyperparameter choices: across all 28 configurations, DisTrust ejects adversaries in every setting where $l \geq 8$ and $r \leq 5$, and false positives can

be driven to zero by reducing l or increasing the initial trust endowment. Under 3-majority, false positives are further suppressed because fault attribution requires committee agreement rather than a single check. The operator’s only meaningful choice is the aggressiveness–accuracy tradeoff, which is a policy decision rather than a tuning problem.

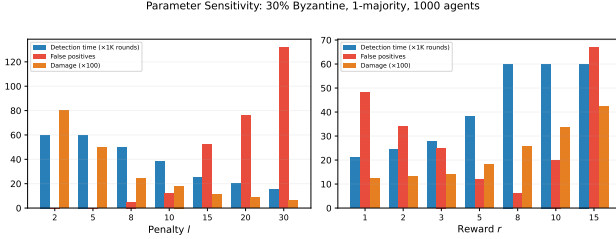


Fig. 8. Parameter sensitivity at 1000 agents, 30% Byzantine, 1-majority. The penalty l is the dominant lever; remaining parameters have moderate to minimal effect.

F. Overhead Analysis

Fig. 9 characterizes coordinator-side overhead. Verification adapts to threat level: 31% of dispatches at 30% adversarial fraction, compared to a fixed 50% for 2x and 67% for 3x replication. DisTrust achieves 692 real tasks per 1000 dispatches, a 38% improvement over 2x replication, and overhead drops further after adversary ejection. Unlike fixed replication, DisTrust’s elevated verification rate is transient: it spikes during the detection phase, drops sharply once adversarial agents are ejected, and stabilizes at a low baseline determined by the decay rate. Measured dispatch latency is 272 μ s mean (642 μ s p99) for 1000 worker processes, scaling linearly from 141 μ s at 10 workers. Per-agent state requires 188 bytes, yielding under 184 KB total for 1000 agents.

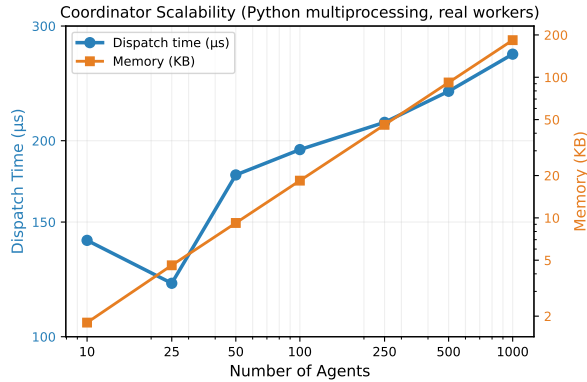


Fig. 9. Coordinator scalability measured in a Python multiprocessing prototype. Dispatch latency increases from 141 μ s to 272 μ s over a 100 $\times$ increase in agent count; memory scales linearly at 188 bytes per agent.

VIII. DRIFT-BASED ANALYSIS OF THE OBSERVED BEHAVIOR

We analyze the trust dynamics through a controlled stochastic-process lens to explain the behavior observed in the

preceding evaluation: we identify the quantities that govern whether honest agents retain trust, whether adversarial agents lose trust, and why the system exhibits a sharp transition near the $N/2$ boundary.

A. Bounded State and Trust Dynamics

Let the system state at time t be $S_t = (T_1(t), \dots, T_n(t), R_t, Q_t, A_t)$, where $T_i(t)$ is agent i ’s trust coins, R_t is the coordinator’s reserve, Q_t is the stored-result pool, and A_t is the active agent set. Trust coins are bounded: $T_i(t) \in [0, C_i]$ for each agent, where C_i is the maximum capacity. Because trust is capped for each agent and the coordinator stores only bounded per-agent state, the dynamics evolve in a bounded state space: trust cannot grow without limit, and agents eventually move toward either sustained positive trust or ejection.

B. Average Trust Drift

A useful approximation is to track the expected change in an agent’s trust over time. Averaged over many task assignments, the trust dynamics can be summarized by the following drift expression:

$$\mathbb{E}[\Delta T_i(t) \mid \mathcal{F}_t] \approx r \cdot P_i^{\text{reward}}(t) - l \cdot P_i^{\text{penalty}}(t) - \lambda/\delta \quad (2)$$

This expression is not intended as an exact closed-form model of every scheduler event. Instead, it isolates the three forces that shape trust over time: reward from successful checking, penalty from disagreement or detection, and background decay. In practice, the relative size of these terms determines whether an agent’s trust tends to rise or fall.

C. Majority Composition and the $N/2$ Boundary

To understand the majority mechanism, we use a mean-field approximation in which a verification committee is treated as a random sample from the active population. Under this approximation, the probability that a committee of size m is adversary-dominated is:

$$\beta_m(\rho_t) = \sum_{k=\lceil m/2 \rceil + 1}^m \binom{m}{k} \rho_t^k (1 - \rho_t)^{m-k} \quad (3)$$

This explains the observed $N/2$ boundary. When $\rho_t < 1/2$, honest-majority committees are more likely, so mismatches are usually attributed correctly and Byzantine agents tend to lose trust. When $\rho_t > 1/2$, the mechanism can invert because adversarial committees become common enough to misattribute fault, which is the transition visible in the Byzantine heatmaps.

D. Interpreting Honest-Agent Stability

When the adversarial fraction stays below one half, an honest agent is usually evaluated by committees that are majority-honest. In that regime, honest agents are rewarded for successful checks and are only occasionally penalized by unlucky committee composition or transient disagreement. As long as the reward rate is large enough relative to false-penalty events and trust decay, honest agents tend to remain in the positive-trust region rather than drifting toward ejection.

This matches the low false-positive rates observed in the experiments below the $N/2$ threshold.

This intuition is captured by the condition

$$r \cdot P_h^{\text{reward}}(t) > l \cdot P_h^{\text{false-penalty}}(t) + \lambda/\delta, \quad (4)$$

which says that honest trust tends to remain stable when reward dominates false penalties and decay on average. This predicts low false-positive behavior in the parameter regimes where honest reward outweighs false-penalty events and decay.

E. Byzantine Ejection and Connection to Experiments

A Byzantine agent that corrupts its real-work outputs creates a recurring source of future penalties: those outputs can later be sampled for verification and judged by an honest-majority committee. Each detected bad output costs the agent l trust coins, so an agent starting with T_0 trust can survive at most $\lceil T_0/l \rceil$ detections before ejection. Omniscience does not change this: the adversary can behave correctly on check work, but cannot prevent honest agents from verifying its corrupted real-work trail. DisTrust is also resilient to dormant-then-attack strategies: continuous decay forces dormant agents to actively earn trust just to maintain their level, and asymmetric penalties ($l > r$) ensure that any subsequent attack depletes accumulated trust faster than it was earned. This explains the three recurring patterns in our experiments: (i) non-colluding faulty agents are ejected fastest because they cannot coordinate committee majorities; (ii) Byzantine agents are reliably ejected below $N/2$ and dominate above it; and (iii) omniscience provides little advantage below $N/2$.

IX. CONCLUSION

We presented DisTrust, a lightweight trust management framework that models trust as a scarce, decaying currency and uses adaptive redundancy to detect and eject faulty and Byzantine agents without requiring assumptions about the structure or content of computed results. We provided a drift-based analysis explaining why honest agents tend to stay while adversarial agents tend to be ejected below the $N/2$ threshold, and validated these predictions empirically against an omniscient adversary. Among all evaluated approaches, DisTrust is capable of ejecting gaming Byzantine adversaries that defeat fixed replication and threshold-based auditing schemes. DisTrust's trust-as-currency design creates an asymmetric game that the adversary cannot win below $N/2$: trust decays continuously, penalties exceed rewards, and trust directly controls computational influence rather than serving as an advisory score. This makes the mechanism enforceable and self-accelerating: adversarial agents are suppressed before ejection and cannot sustain their position through any strategy, including building trust during a dormant phase before attacking. The framework achieves false positives under 2% at moderate initial trust, imposes lower overhead than other tested methods while providing stronger guarantees, and requires negligible coordinator-side resources. DisTrust is problem-agnostic and can be used as a pluggable layer, making it applicable to autonomous system deployments including robotic

fleets, sensor networks, and distributed edge computing. When used as an augmentation layer, DisTrust is strictly beneficial: it improves any underlying defense by providing runtime adaptive resilience that converges the system to attacker-free output quality with negligible coordinator-side cost. Future work includes decentralizing the coordinator, extending to non-deterministic workloads, and evaluation on physical multi-robot testbeds.

ACKNOWLEDGMENT

This material is based in part upon work supported by the National Science Foundation under Grant Numbers CNS-2333487 and CNS-2140139, and the Army Research Office under Contract W911NF-2020-221. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes. The views and conclusions contained herein are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

- [1] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: An experiment in public-resource computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, 2002.
- [2] D. P. Anderson, "BOINC: A system for public-resource computing and storage," in *5th IEEE/ACM Int. Workshop on Grid Computing*, 2004, pp. 4–10.
- [3] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Advances in Cryptology, CRYPTO 2010*, vol. 6223, pp. 465–482, 2010.
- [4] L. Chen, H. Wang, Z. Charles, and D. Papailiopoulos, "Draco: Byzantine-resilient distributed training via redundant gradients," in *Proc. ICML*, 2018, pp. 903–912.
- [5] S. Rajput, H. Wang, Z. Charles, and D. Papailiopoulos, "Detox: A redundancy-based framework for faster and more robust gradient aggregation," in *Advances in NeurIPS*, vol. 32, 2019.
- [6] S. Liu, N. Gupta, and N. H. Vaidya, "Approximate Byzantine fault-tolerance in distributed optimization," in *Proc. ACM PODC*, 2021, pp. 379–389.
- [7] G. Damaskinos, E. M. El Mhamdi, R. Guerraoui, R. Patra, and M. Taziki, "Asynchronous Byzantine Machine Learning (the case of SGD)," in *Proc. ICML*, vol. 80, 2018, pp. 1145–1154.
- [8] K. Garlichs, A. Willecke, M. Wegner, and L. C. Wolf, "TriP: Misbehavior detection for dynamic platoons using trust," in *Proc. IEEE 22nd Int. Conf. on Intelligent Transportation Systems (ITSC)*, 2019.
- [9] A. De Paola and A. Tamburo, "Reputation management in distributed systems," in *3rd Int. Symp. on Communications, Control and Signal Processing*, 2008, pp. 666–670.
- [10] R. Pinyol and J. Sabater-Mir, "Computational trust and reputation models for open multi-agent systems: A review," *Artificial Intelligence Review*, vol. 40, no. 1, pp. 1–25, 2013.
- [11] J. Regatti, H. Chen, and A. Gupta, "Byzantine resilience with reputation scores," in *58th Annual Allerton Conf.*, 2022, pp. 1–8.
- [12] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Proc. ICML*, vol. 80, 2018, pp. 5650–5659.
- [13] S. Gyawali, Y. Qian, and R. Q. Hu, "Machine learning and reputation based misbehavior detection in vehicular communication networks," *IEEE Trans. Vehicular Technology*, vol. 69, no. 8, pp. 8871–8885, 2020.
- [14] L. Wan, Y. Cao, C. Bi, Z. Liu, J. Ma, and Y. Ren, "PBRU: Privacy-preserving and blockchain-assisted reputation updating with malicious detection for cloud-supported vehicular networks," *IEEE Trans. Information Forensics and Security*, vol. 20, pp. 13026–13040, 2025.
- [15] R. Guerraoui, S. Rouault, et al., "The hidden vulnerability of distributed learning in Byzantium," in *Proc. ICML*, 2018, pp. 3521–3530.
- [16] V. Buterin and V. Griffith, "Casper the Friendly Finality Gadget," *arXiv preprint arXiv:1710.09437*, 2017.