Byzantine-Inspired Cryptocurrency Fraud Detection Through ML-Anchored Multi-Agent Consensus

Neil Chandra[1]

Received May 26, 2026
Accepted July 7, 2026
Electronic access July 31, 2026

Cryptocurrency fraud detection systems relying on single supervised learning models are vulnerable to systematic errors that propagate directly to the final prediction. This study introduces ML-Anchored Supermajority Consensus, a Byzantine-inspired algorithm combining a Random Forest baseline with four LLM-based agents under a confidence-gated supermajority voting mechanism. The baseline serves as a trusted anchor outside the voting set, overridable only when a supermajority of agents disagree and baseline confidence falls below a specified threshold. Evaluation used 10,000 transactions per condition from the Bitcoin Elliptic and BitcoinHeist datasets across three fraud rate conditions (5%, 10%, 20%), with 5-fold cross-validation and two adversarial models, naive vote-flip and mechanism-aware targeted at four agent compromise levels (0% – 75%). The multi-agent system improved recall over the Random Forest baseline by approximately 10 percentage points across all fraud rate conditions, with statistically significant results well below $\alpha = 0.05$ and large effect sizes. All recall gains are attributable to BitcoinHeist; the supermajority override was never triggered on Bitcoin Elliptic. Precision decreased substantially across all conditions, reflecting the conservative bias of the asymmetric override design; under an asymmetric cost model (Cost = FP + 20×FN), the false negative reduction could yield lower aggregate cost. Accuracy remained above 93% under both adversary types at all compromise levels. Eight formal results, four algorithmic properties by construction and four derived behavioral consequences were confirmed with zero violations. These findings represent empirical robustness observations under the tested adversarial conditions rather than formal Byzantine fault tolerance guarantees. Limitations include agent architectural homogeneity, absence of runtime benchmarking, and a two-dataset evaluation scope.

Introduction

Cryptocurrency fraud represents a rapidly escalating threat to digital financial systems, with economic losses growing substantially as blockchain adoption expands. Detection of fraudulent transactions is complicated by the pseudonymous nature of blockchain addresses and the similarity of transaction patterns between legitimate and malicious activity, motivating the development of automated detection systems capable of operating at scale.

Existing approaches rely on supervised machine learning and graph-based methods applied to blockchain transaction data. Graph convolutional networks on Bitcoin transaction graphs have outperformed traditional feature engineering for illicit transaction detection[1], while topological data analysis has demonstrated meaningful fraud signals in local Bitcoin graph structure, yielding the BitcoinHeist dataset[2]. Individual classifiers including Random Forest and XGBoost have been applied directly to Bitcoin transaction classification[3], with machine learning pipelines incorporating pre-training and deanonymization demonstrating strong performance on the Elliptic dataset[4]. Ensemble approaches consistently outperform individual classifiers, both on the Elliptic dataset[5] and on Ethereum using symmetric aggregation methods including hard voting, stacking, and boosting[6–9], yielding consistent gains over single classifiers[10–13]. Hybrid ensemble approaches combining bagging and boosting methods have been proposed specifically for Bitcoin fraud detection, demonstrating improved performance over single-model and single-type ensemble baselines[14]. Graph learning approaches have further shown strong performance on blockchain phishing and anomaly detection tasks, with heterogeneous representations outperforming homogeneous ones[15,16]. Despite this progress, single-model approaches rely on a single decision component whose errors propagate directly to the system output, while the ensemble approaches reviewed here employ symmetric aggregation in which all classifiers contribute equally, without a designated trusted baseline or confidence-gated override layer.

Byzantine fault tolerance (BFT), first formalized by Lamport, Shostak and Pease[17] and extended to practical distributed systems by Castro and Liskov[18], establishes that consensus among n components can tolerate at most $f < n/3$ arbitrary failures in a fully distributed setting requiring decentralized consensus, replicated state machines, and authenticated communication, an architecture fundamentally distinct from

[1] *Del Norte High School, California, USA*

the present work, which draws conceptual motivation from BFT without implementing distributed consensus. Consensus in networked multi-agent systems has been studied through convergence conditions for distributed algorithms[19,20] and belief-updating models[21].

Recent peer-reviewed work has begun applying BFT-inspired reasoning to LLM-based multi-agent systems. Zheng et al. propose CP-WBFT, a confidence-weighted BFT-inspired consensus mechanism demonstrating empirical robustness under high Byzantine fault rates[22]. Chen et al. address Byzantine robustness through blockchain-based proof-of-thought consensus, identifying agent collusion as a primary vulnerability[23]. Huang et al. find that hierarchical configurations exhibit substantially lower performance degradation than flat or linear structures under simulated fault conditions, framing results as empirical robustness observations rather than formal guarantees[24]. Liang et al. demonstrate that structured multi-agent debate mitigates the Degeneration-of-Thought problem, where single LLMs maintain confident but incorrect conclusions through self-reflection[25]. Chen, Saha and Bansal show that confidence-weighted voting among architecturally diverse LLM agents outperforms same-model multi-agent baselines, with agent diversity identified as a critical driver of consensus quality[26]. Together these studies establish that empirical robustness under controlled fault conditions, rather than formal BFT guarantees, is the appropriate characterization for this class of system.

Prior work has not examined the combination of a trusted ML anchor with BFT-inspired LLM multi-agent consensus specifically for cryptocurrency fraud detection. This research introduces ML-Anchored Supermajority Consensus, a Byzantine-inspired algorithm combining a supervised machine learning baseline with four LLM-based agents under a confidence-gated supermajority voting mechanism, motivated by the structural limitations of single-model and symmetric-ensemble approaches identified above. Two formal hypotheses guide this investigation: the performance hypothesis (H_1) predicts statistically significant recall improvement over the Random Forest baseline ($\Delta r > 5$ percentage points, $p < 0.05$, paired t-test across 5-fold cross-validation), and the resilience hypothesis (H_2) predicts maintenance of $\geq 90\%$ baseline accuracy under agent compromise at rates up to 75% under both naive and mechanism-aware adversarial conditions.

The algorithm's behavior is characterized through eight formal results: four properties following directly from the algorithm's definition by construction (monotonicity, override conditions, confidence-conditional override, and decision regions), three derived consequences characterizing non-trivial aspects of aggregate system behavior (recall lower bound, asymmetric escalation, and conservative bias), and a computational complexity bound under idealized parallel execution. Evaluation used 10,000 cryptocurrency transactions per fraud rate condition sampled from the Bitcoin Elliptic[1] and BitcoinHeist[2] datasets via stratified random sampling across three fraud rate conditions (5%, 10%, and 20%), with 5-fold cross-validation, paired t-tests, and Byzantine compromise experiments at 0%, 25%, 50%, and 75% agent compromise levels under both naive vote-flip and mechanism-aware targeted adversary models.

Several limitations constrain generalizability. The 10,000-transaction sample size per fraud rate condition may limit detection of rare fraud patterns. The confidence-gating mechanism introduces a vulnerability surface where adversaries targeting the baseline's confidence estimate could suppress override consideration; while the targeted adversary evaluated coordinated override exploitation through false positive inflation, direct manipulation of baseline confidence was not assessed. Production deployment would require self-hosted LLMs rather than commercial APIs to maintain data privacy compliance. All four agents are instances of the same underlying model (GPT-4o-mini); prior work identifies agent architectural diversity as a critical driver of consensus quality[26], and this homogeneity may limit the diversity of reasoning the consensus mechanism can draw on.

Methods

Research Design

This is an experimental study combining supervised machine learning with LLM-based multi-agent consensus mechanisms to develop a Byzantine-inspired algorithm for cryptocurrency fraud detection. The study employs a 5-fold cross-validation design to evaluate algorithm performance on binary classification of cryptocurrency transactions as fraudulent or legitimate. The experimental design includes systematic manipulation of agent behavior to assess empirical robustness under adversarial conditions.

Dataset

The study employed 10,000 cryptocurrency transactions obtained through stratified random sampling from two publicly available Kaggle datasets: Bitcoin Elliptic and BitcoinHeist. All datasets consist of transactions with confirmed fraud or legitimate classifications, with unlabeled transactions in Bitcoin Elliptic excluded during preprocessing. To assess system performance across varying fraud prevalence, evaluation was conducted under three fraud rate conditions (5%, 10%, and 20%) through stratified sampling, yielding 10,000 transactions per condition.

Bitcoin Elliptic Dataset

The Bitcoin Elliptic dataset represents Bitcoin transaction graph data collected by Elliptic Co. in collaboration with the

MIT-IBM Watson AI Lab. Each observation corresponds to a single Bitcoin transaction represented as a node in the transaction graph. The dataset contains 166 anonymized features organized into two categories. Local transaction features (features 1–94) encode properties intrinsic to each transaction: feature 1 represents the time step (1–49, corresponding to two-week intervals), while features 2–94 capture transaction properties including total input/output Bitcoin values, transaction fees, number of input/output addresses, and statistical aggregates of transaction amounts. Aggregated neighborhood features (features 95–166) represent one-hop statistics (maximum, minimum, standard deviation, and correlation coefficients of neighbor transactions) computed over each transaction's neighboring nodes in the graph, capturing network connectivity patterns.

Transactions are labeled as illicit (class 1), licit (class 2), or unknown. Only transactions with confirmed labels were retained; unlabeled transactions were excluded.

BitcoinHeist Dataset

The BitcoinHeist dataset[2] catalogs Bitcoin addresses associated with ransomware campaigns. Each observation represents a Bitcoin address labeled with a specific ransomware family (Locky, CryptoLocker, WannaCry) or "white" (not known to be ransomware). The dataset contains 8 base features: year (calendar year), day (day of year), length (mixing-round depth), weight (a merge-accumulation feature capturing what fraction of transaction outputs accumulate at the address through chains of merging transactions), count (the merge-pattern transaction count), looped (the count of split-then-merge laundering patterns terminating at the address), neighbors (number of distinct addresses transacted with), and income (total satoshis received). All ransomware family labels were collapsed into ransomware (class 1); white addresses became class 0.

Data Collection

Dataset Acquisition

All datasets were downloaded from Kaggle. Bitcoin Elliptic provided three CSVs (elliptic_txs_features.csv, elliptic_txs_edgelist.csv and elliptic_txs_classes.csv) that were loaded separately and merged on transaction ID. BitcoinHeist provided a single CSV.

Data pre-processing

All datasets underwent a shared pre-processing template, with column selection and feature engineering adapted to each dataset's schema. Missing value imputation was performed in two steps: first, infinite values (np.inf, -np.inf) arising from division operations were converted to NaN using pandas replace function; second, all missing values were imputed using column-wise median imputation, where each feature's missing values were filled with that feature's median computed from non-missing observations. Median imputation was selected over mean imputation to minimize outlier influence, particularly important for cryptocurrency data exhibiting heavy-tailed distributions.

For Bitcoin Elliptic, unlabeled observation removal filtered out transactions with class "unknown," retaining only rows where class was "1" (illicit) or "2" (licit). As a defensive measure, any object-typed columns remaining in the feature matrix were passed through sklearn.preprocessing.LabelEncoder after filling missing categories with the string "Unknown," though in practice this fallback did not apply once identifier columns were removed.

Feature Engineering

Feature engineering augmented raw datasets with derived variables capturing additional predictive signals. Bitcoin Elliptic received no engineering as the dataset was pre-engineered by Elliptic Co. with 166 comprehensive features.

BitcoinHeist received 8 derived features (16 total): income_per_tx = income/(count+1) computes average satoshis per transaction; income_per_neighbor = income/(neighbors+1) normalizes income by connectivity; weight_length_ratio = weight/(length+1) expresses merge-accumulation strength relative to mixing-round depth; count_per_neighbor = count/(neighbors+1) measures merge-pattern transaction count per connected address; log_income = log(1+income) and log_weight = log(1+weight) compress heavy-tailed distributions; count_x_length = count$\times$length captures the combination of merge-pattern transaction count and mixing-round depth; neighbors_x_looped = neighbors$\times$looped captures the interaction between neighborhood size and the count of split-then-merge laundering patterns terminating at the address.

Feature Scaling

Each dataset received its own StandardScaler (scikit-learn), fit exclusively on the training split to prevent data leakage and persisted to disk (as {dataset}_scaler.pkl) for reuse at inference. Features were transformed to zero mean and unit variance. At inference time, each sample was scaled using the scaler corresponding to its source dataset, since the two datasets have heterogeneous feature schemas: Bitcoin Elliptic's anonymized graph features and BitcoinHeist's graph-derived ransomware features.

Data Partitioning and Leakage Control

Each dataset was divided into three disjoint partitions before any modeling: a training partition used to fit the Random Forest classifier and feature scaler, a validation partition used solely to select the decision threshold, and an evaluation partition reserved exclusively for performance measurement. De-

cision thresholds were selected by maximizing F1 on the validation partition only; the evaluation partition played no role in fitting, scaling, feature construction, or threshold selection. BitcoinHeist features were computed independently per transaction from that transaction's own attributes, introducing no dependence across partition boundaries. The two forests, their scalers, and their thresholds were fixed after training and never modified during evaluation. The five cross-validation folds operated exclusively on the evaluation partition to quantify variance in consensus metrics; no fold was used to refit any model component. Record counts per dataset are reported in Table 1.

Table 1 Data partitioning

Dataset	Training	Validation	Evaluation	Total
Bitcoin Elliptic	32,594	4,656	9,314	46,564
Bitcoin Heist	2,041,687	291,670	583,340	2,916,697

Variables and Measurements

Independent Variables

Independent variables consist of the transaction-level features described in the Dataset section: 166 features for Bitcoin Elliptic and 16 features for BitcoinHeist.

Dependent Variable

The dependent variable is binary fraud classification where 0 indicates legitimate transactions and 1 indicates fraudulent transactions. For Bitcoin Elliptic, fraud labels distinguish illicit (class 1) from licit (class 2, recoded to 0) transactions. For BitcoinHeist, fraud labels distinguish ransomware addresses (class 1) from white addresses (class 0).

Algorithm Outputs

The algorithm produces multiple intermediate and final outputs. $\hat{y}_{ML}$ and p_{ML} represent the baseline Random Forest[6] prediction and its associated confidence, respectively. Agent votes $v_1, v_2, v_3, v_4 \in \{0,1\}$ represent the four LLM-based agents' fraud classifications, summed as $s = \sum v_i$. The override signal $o \in \{0,1\}$ activates when $\hat{y}_{ML} = 0$, $s \geq k$, and $p_{ML} < \tau$. The final prediction $\hat{y}_{final} \in \{0,1\}$ represents the algorithm's consensus output.

Evaluation Metrics

Model performance is quantified using standard binary classification metrics. Recall (sensitivity, true positive rate) measures the proportion of actual fraud cases correctly identified: Recall $= TP/(TP+FN)$. Precision measures the proportion of predicted fraud cases that are fraudulent: Precision $= TP/(TP+FP)$. Accuracy measures overall classification correctness: Accuracy $= (TP+TN)/(TP+TN+FP+FN)$. F1-score represents the harmonic mean of precision and recall: F1 $= 2 \times (\text{Precision} \times \text{Recall})/(\text{Precision} + \text{Recall})$. False negative rate quantifies the proportion of fraud cases incorrectly classified as legitimate: FNR $= FN/(TP+FN)$. False positive rate quantifies the proportion of legitimate transactions incorrectly classified as fraud: FPR $= FP/(FP+TN)$. For adversarial robustness evaluation, an asymmetric cost function is applied: Cost $= FP + 20 \times FN$, reflecting the operational assumption that a missed fraud case incurs twenty times the cost of a false alert, the former representing direct uninvestigated financial loss and the latter representing investigator review time. All metrics are computed per fold and aggregated across 5-fold cross-validation.

Procedure

Algorithm Architecture

The ML-Anchored Supermajority Consensus algorithm combines a supervised machine learning baseline with four specialized LLM-based agents operating under a supermajority consensus protocol. Two primary components comprise the architecture alongside the consensus layer: two Random Forest classifiers (one per dataset) providing baseline predictions, and four LLM-based agents analyzing transactions through distinct analytical frameworks. A confidence-gated supermajority consensus mechanism aggregates agent votes to determine whether the baseline prediction is escalated.

The consensus mechanism implements asymmetric voting where the baseline prediction serves as default, overridable only when (1) the baseline predicts SAFE, (2) at least k of n agents vote for an alternative (FRAUD) classification, and (3) the baseline's own confidence in its SAFE prediction falls below threshold τ. This asymmetric design contrasts with traditional ensemble methods such as AdaBoost[7] and Random Forest[6] which employ symmetric aggregation where all predictors contribute equally through weighted voting or averaging.

Mathematical Formulation

The algorithm's output is formally defined as

$$\hat{y}_{final} = \hat{y}_{ML} \vee o \tag{1}$$

where $\hat{y}_{ML} \in \{0,1\}$ represents the Random Forest baseline prediction, $o \in \{0,1\}$ denotes the override signal, and $\vee$ represents logical OR. The baseline classifier f additionally produces a confidence score $p_{ML} \in [0,1]$: $(\hat{y}_{ML}, p_{ML}) \leftarrow f(x)$.

Each of the n agents produces a binary vote $v_i \in \{0,1\}$ for whether the transaction is fraudulent. Let $s = \sum_{i=1}^{n} v_i$ denote the number of agents voting FRAUD.

The override signal activates ($o = 1$) when three conditions hold simultaneously: (1) the ML baseline predicts SAFE ($\hat{y}_{ML} = 0$), (2) a supermajority of agents vote FRAUD ($s \geq k$),

and (3) the baseline's confidence in its SAFE prediction falls below threshold τ ($p_{ML} < \tau$):

$$o = \mathbb{1}\left[\hat{y}_{\text{ML}} = 0 \wedge s \geq k \wedge p_{\text{ML}} < \tau\right]. \quad (2)$$

where $\mathbb{1}[\cdot]$ is the indicator function. Since $\hat{y}_{final} = \hat{y}_{ML} \vee o$, the override signal affects the final prediction only when $\hat{y}_{ML} = 0$; a FRAUD baseline prediction yields $\hat{y}_{final} = 1$ irrespective of o and cannot be reversed by agent votes (Theorem 1).

The implemented parameters are $\tau = 0.7$, $k = 3$, and $n = 4$. $\tau = 0.7$ restricts override consideration to cases where the baseline's confidence in its SAFE prediction is sufficiently low to warrant agent input; $k = 3$ of $n = 4$ requires agreement among 75% of agents, exceeding simple majority.

Theoretical Properties

The algorithm's behavior admits several formal properties, which fall into two categories. Theorem 1 (Monotonicity Guarantee), Theorem 2 (Override Necessity), Lemma 1 (Confidence-Conditional Override), and Theorem 4 (Decision Regions) follow directly from the algorithm's definition and serve primarily to make explicit guarantees that are implicit in the override mechanism's construction. Corollary 1 (Recall Lower Bound), Theorem 3 (Asymmetric Escalation), and Corollary 2 (Conservative Bias) are derived consequences obtained by combining these definitional properties, and characterize non-trivial aspects of the algorithm's aggregate behavior, specifically, its effect on recall, the direction in which verdicts can be revised, and the resulting precision-recall tradeoff. Theorem 5 addresses runtime complexity under an idealized parallel-execution assumption.

Theorem 1 (Monotonicity): If the ML baseline classifies a transaction as fraudulent ($\hat{y}_{ML} = 1$), the final verdict will always be fraudulent ($\hat{y}_{final} = 1$), regardless of agent votes. Fraud verdicts cannot be dismissed.

Corollary 1 (Recall Lower Bound): The algorithm's recall satisfies $\text{Recall}_{final} \geq \text{Recall}_{ML}$, establishing that performance never degrades below the ML baseline. The ML anchor acts as a performance floor with agent consensus providing only upward corrections.

Theorem 2 (Override Necessity): The override mechanism activates if and only if three conditions are simultaneously satisfied: (1) ML baseline predicts SAFE ($\hat{y}_{ML} = 0$), (2) supermajority of agents predict FRAUD ($s \geq k$), and (3) ML baseline exhibits low confidence ($p_{ML} < \tau$). Override occurs only when justified by both agent agreement and baseline uncertainty.

Lemma 1 (Confidence Respect): If the ML baseline predicts SAFE with high confidence ($p_{ML} \geq \tau$), override cannot activate regardless of agent votes. High-confidence baseline predictions are protected from agent override.

Theorem 3 (Asymmetric Escalation): The algorithm can escalate verdicts from SAFE to FRAUD via override but cannot de-escalate verdicts from FRAUD to SAFE. This asymmetry ensures that agent votes can only escalate verdicts from SAFE to FRAUD, not reverse FRAUD verdicts to SAFE.

Corollary 2 (Conservative Bias): The algorithm exhibits systematic conservative bias favoring false positives over false negatives. False negatives can only decrease via escalation, while false positives can increase, trading increased FP for decreased FN.

Theorem 4 (Decision Completeness): The algorithm partitions the input space into three disjoint decision regions that exhaustively classify all transactions: Monotonic Fraud ($\hat{y}_{ML} = 1$), Override-Escalated Fraud ($\hat{y}_{ML} = 0 \wedge s \geq k \wedge p_{ML} < \tau$), and Approved Safe ($\hat{y}_{ML} = 0 \wedge (s < k \vee p_{ML} \geq \tau)$).

Theorem 5 (Parallel Aggregation Complexity): If agent predictions are computed in parallel, the algorithm's agent-aggregation step has time complexity $O(T)$, where T is the maximum time required by any single agent, compared to $O(nT)$ under sequential execution. This bound assumes idealized parallel execution and does not account for LLM inference latency or API overhead.

All eight formal properties were confirmed to hold across the 10,000-transaction evaluation dataset, consistent with their derivation from the algorithm's construction.

ML Baseline Algorithm Selection

Three supervised learning algorithms were evaluated as candidates for the ML baseline: Random Forest[6], Gradient Boosting, and Logistic Regression. Performance was compared across accuracy, F1-score, and ROC AUC metrics.

Table 2 ML Comparison

	Random Forest	Gradient Boosting	Logistic Regression
Accuracy	0.99	0.98	0.89
F1	0.94	0.90	0.63
ROC AUC	0.99	0.99	0.96

Random Forest demonstrated superior performance across all metrics, particularly in F1-score which balances precision and recall. Based on these results, Random Forest was selected as the ML baseline.

Random Forest Training

Two separate Random Forest classifiers were trained, one per dataset (Bitcoin Elliptic and BitcoinHeist), using scikit-learn's RandomForestClassifier. Each Random Forest was fit on a class-balanced subsample of its training partition, 15,905 records for Bitcoin Elliptic and 144,945 for BitcoinHeist, obtained by majority-class undersampling to a 20% fraud target. The decision threshold for each forest was selected by maximizing F1 on its validation partition, yielding 0.440 for Bitcoin Elliptic and 0.695 for BitcoinHeist. Each StandardScaler was estimated on its training partition alone. The two

forests, their scalers, and their thresholds were frozen after training and were not modified during evaluation. Full partitioning methodology and record counts are described in the Data Partitioning and Leakage Control section and reported in Table 1.

Cross-Validation Protocol

The evaluation sample of 5,000 transactions per dataset (10,000 pooled) was drawn exclusively from the withheld evaluation partitions under three fraud rate conditions (5%, 10%, and 20%) through stratified sampling. For each test fold transaction, the algorithm routed it to the appropriate pretrained Random Forest baseline, obtained the baseline prediction and confidence score, invoked the four LLM agents, and applied the supermajority consensus mechanism. This procedure was repeated across all 5 folds, yielding 5 independent performance measurements per fraud rate condition.

LLM Agent Design

Four specialized LLM-based agents were implemented using OpenAI GPT-4o-mini, each configured with distinct analytical roles through prompt differentiation. All four agents share the same underlying model; diversity was induced through role-specific prompting rather than architectural heterogeneity, which represents a limitation of the current implementation. System parameters were set to $\tau = 0.7$, $k = 3$, and $n = 4$. Abbreviated agent prompts are reproduced below for reproducibility:

Fraud Analyst: "You are a fraud analyst. The Random Forest prediction is your primary signal and is accurate on held-out data. Agree with it unless the feature values show a strong, specific anomaly. Start your reply with exactly 'VOTE: HIGH_RISK (X% confidence)' or 'VOTE: LOW_RISK (X% confidence)'."

Security Watchdog: "You are a security watchdog looking for anomalies the baseline may miss. Treat the ML prediction as the default and only diverge on concrete red flags such as extreme outliers or impossible values. Do not flag a transaction merely because the features are unfamiliar. Start your reply with exactly 'VOTE: HIGH_RISK (X% confidence)' or 'VOTE: LOW_RISK (X% confidence)'."

Adversarial Analyst: "You think like an attacker trying to evade detection. Ask whether the features could be a deliberate manipulation aimed at a low fraud score. Most transactions are legitimate, so do not be paranoid; vote HIGH_RISK only on specific signs of evasion. Start your reply with exactly 'VOTE: HIGH_RISK (X% confidence)' or 'VOTE: LOW_RISK (X% confidence)'."

Consensus Builder: "You synthesize the ML prediction with the feature evidence. Maintain agreement with the baseline unless the evidence against it is overwhelming across several dimensions. Start your reply with exactly 'VOTE: HIGH_RISK (X% confidence)' or 'VOTE: LOW_RISK (X% confidence)'."

Agent responses were parsed to extract binary predictions (HIGH_RISK $\rightarrow$ 1, LOW_RISK $\rightarrow$ 0). Individual agent confidence scores expressed in the response format were not used in the consensus mechanism; only binary votes contributed to the supermajority count s. Responses failing to conform to the specified output format were treated as abstentions; in such cases n was reduced accordingly, and the $k = 3$ threshold was applied to the reduced panel.

Consensus Mechanism Implementation

The consensus mechanism follows the mathematical formulation described in the Procedure section. If the baseline predicts FRAUD ($\hat{y}_{ML} = 1$), the final prediction is FRAUD regardless of agent votes (Theorem 1). If the baseline predicts SAFE ($\hat{y}_{ML} = 0$), override activates only if $s \geq k$ and $p_{ML} < \tau$, as defined in the override condition $o = \mathbb{1}[\hat{y}_{ML} = 0 \wedge s \geq k \wedge p_{ML} < \tau]$.

Byzantine Testing Protocol

Algorithm evaluation consisted of two experimental trials conducted across the 5-fold cross-validation framework.

Trial 1 (Normal Conditions): The ML baseline Random Forest and the complete multi-agent consensus system were evaluated on identical test sets across all 5 folds across three fraud rate conditions (5%, 10%, and 20%). Performance metrics including accuracy, precision, recall, and F1-score were calculated for both systems. Paired t-tests were conducted to assess statistical significance of recall differences between the baseline and consensus algorithm across folds, with effect sizes reported as Cohen's d.

Trial 2 (Adversarial Conditions): The multi-agent consensus system was evaluated under two adversarial models at four compromise levels (0%, 25%, 50%, and 75% of agents). The first, a naive vote-flip adversary, simulated Byzantine behavior by forcing compromised agents to deterministically invert their votes on all transactions regardless of features or ML baseline output. The second, a mechanism-aware targeted adversary, simulated a more realistic threat by directing compromised agents to cast coordinated high-confidence FRAUD votes on legitimate transactions, specifically targeting the supermajority override mechanism to inflate false positives. Accuracy, precision, recall, FPR, FNR, and cost were measured at each compromise level across all three fraud rate conditions. Agent confidence calibration under adversarial conditions was not assessed and represents a limitation of the adversarial evaluation.

Data Analysis

Performance metrics including accuracy, precision, recall, and F1-score were calculated for both systems on identical test

sets across folds. Paired t-tests with significance threshold $\alpha = 0.05$ were conducted on recall as the primary outcome measure, with effect sizes reported as Cohen's d alongside 95% confidence intervals. The null hypothesis posited no difference in recall between systems, while the alternative hypothesis posited superior recall for the consensus algorithm.

Adversarial robustness analysis quantified system behavior under two adversarial models by measuring accuracy, precision, recall, FPR, FNR, and cost at each compromise level (0%, 25%, 50%, 75%) across all three fraud rate conditions. Agent confidence calibration under adversarial conditions was not assessed and represents a limitation of the adversarial evaluation.

Theorem validation employed exhaustive verification across each 10,000-transaction evaluation condition. For each formal property, every transaction classification was checked against the formal statement to identify violations. A property was considered confirmed if zero violations occurred across all evaluated transactions, consistent with its derivation from the algorithm's construction.

Ethical Considerations

This research utilized publicly available blockchain transaction data from two established datasets. All data consists of pseudonymous blockchain addresses containing no personally identifiable information. No human subjects were involved in data collection or algorithm evaluation. The datasets represent aggregated blockchain network activity, and this study did not require institutional review board approval.

Results

System Performance Comparison

The ML-Anchored Supermajority Consensus algorithm was evaluated against the Random Forest baseline across three fraud rate conditions (5%, 10%, and 20%), using $n = 5{,}000$ transactions per dataset ($n = 10{,}000$ pooled per condition). Full classification metrics are reported in Table 3.

Table 3 Pooled Classification Performance: ML Baseline vs. Multi-Agent Consensus

	ML Baseline			Multi-Agent		
Fraud rate	5%	10%	20%	5%	10%	20%
Accuracy	97.96%	96.32%	92.97%	96.73%	95.66%	93.41%
Precision	91.34%	95.66%	97.78%	64.74%	79.36%	88.87%
Recall	65.40%	66.20%	64.64%	76.00%	76.50%	74.86%
F1 score	76.22	78.25	77.83	69.92	77.90	81.26

The multi-agent system improved recall over the ML baseline by approximately 10 percentage points across all fraud rate conditions, with a corresponding reduction in false negatives. The precision gap between systems narrowed as fraud prevalence increased, partly due to the shift in class composition: higher fraud rates reduce the pool of legitimate transactions available to generate false positives. Under the asymmetric cost model (Cost = FP + 20×FN, defined in Evaluation Metrics), the reduction in false negatives could result in lower aggregate cost at all fraud rate conditions despite the increase in false positives.

Per-Dataset Analysis

Per-dataset results reveal an asymmetric contribution pattern between Bitcoin Elliptic and Bitcoin Heist (Table 4).

Table 4 Per-Dataset Performance by Fraud Rate

	ML Baseline			Multi-Agent		
Fraud rate	5%	10%	20%	5%	10%	20%
Bitcoin Elliptic						
Precision	97.03%	98.50%	99.28%	97.03%	98.50%	99.28%
Recall	91.60%	92.00%	90.87%	91.60%	92.00%	90.87%
F1 score	94.24	95.14	94.89	94.24	95.14	94.89
Bitcoin Heist						
Precision	80.33%	89.78%	94.88%	43.02%	61.37%	77.71%
Recall	39.2%	40.40%	40.80%	60.40%	61.00%	60.30%
F1 score	52.69	55.72	57.06	50.25	61.18	67.91

On Bitcoin Elliptic, outcomes were identical across both systems at all fraud rate conditions, indicating the supermajority override was never triggered. This is consistent with the high ML baseline recall on this dataset, suggesting the Random Forest anchor classified transactions with sufficient confidence to preclude consensus intervention. All recall gains and precision costs in the pooled results are therefore attributable to Bitcoin Heist, where the supermajority override improved recall by approximately 20 percentage points across all fraud rate conditions, at the cost of increased false positives. Under the asymmetric cost model described above, this reduction in false negatives could result in lower aggregate cost.

Statistical Analysis

Paired t-tests comparing per-fold recall across the two systems yielded statistically significant results well below $\alpha = 0.05$ at all fraud rate conditions, with large effect sizes throughout.

Table 5 Statistical Comparison of Recall (Multi-Agent vs. ML Baseline)

Fraud rate	Mean Recall Diff	95% CI	Cohen's d	t-statistic	p-value
5%	+10.60	[9.133, 12.067]	6.335	14.165	0.00014
10%	+10.30	[9.391, 11.209]	9.934	22.214	2.0×10^{-5}
20%	+10.22	[9.889, 10.543]	27.374	61.210	< 0.001

Confidence intervals were narrow at all fraud rate conditions, confirming the consistency of the recall improvement across cross-validation folds. The increasing Cohen's d across fraud rates reflects decreasing within-condition variance rather than a larger absolute effect, as the mean recall difference remained stable across all three conditions. The p-value at the 20% fraud rate condition approached the limits of floating-point precision in the test implementation and is reported conservatively as < 0.001.

Inter-Agent Agreement

Fleiss' kappa increased with fraud prevalence (0.289, 0.334, and 0.407 at 5%, 10%, and 20% respectively), indicating fair to moderate inter-agent agreement that strengthened as fraud density increased. Pairwise analysis revealed meaningful heterogeneity: the Security Watchdog and Adversarial Analyst showed the highest mutual agreement ($\kappa = 0.69$–0.73), while the Consensus Builder aligned most closely with the ML baseline ($\kappa = 0.98$–0.99) and least with the other agents ($\kappa = 0.06$–0.28). Figure 1 illustrates pairwise and agent-vs-ML agreement at the 20% fraud rate condition.

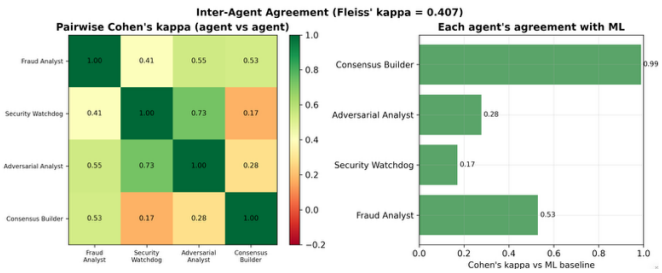

Figure. 1 Pairwise and Agent vs. ML agreement at 20% Fraud rate

Adversarial Robustness

Table 6 Performance Under Adversarial Conditions: Vote-Flip

FR (%)	Compromise (%)	Precision	Accuracy	Recall	FPR	FNR	Cost
5%	0	64.7%	96.7%	76.0%	2.2%	24%	2607
	25	75.0%	97.3%	71.0%	1.2%	29%	3050
	50	91.0%	98.0%	66.0%	0.3%	34%	3400
	75	90.8%	97.9%	65.4%	0.3%	34.6%	3493
10%	0	79.4%	95.7%	76.5%	2.2%	23.5%	4899
	25	87.0%	96.1%	71.0%	1.1%	29.0%	6000
	50	95.5%	96.3%	66.3%	0.4%	33.8%	6791
	75	95.5%	96.3%	66.2%	0.3%	33.8%	6791
20%	0	88.9%	93.4%	74.9%	2.2%	25.1%	9779
	25	93.5%	93.3%	70.0%	1.1%	30.0%	11900
	50	97.5%	93.0%	64.6%	0.4%	35.0%	13531
	75	97.5%	92.9%	64.6%	0.4%	35.4%	13531

Under the vote-flip adversary, precision increased and recall decreased monotonically with agent compromise across all fraud rate conditions, as compromised agents suppressed override activations and pushed the system toward the ML baseline; cost increased monotonically as a result, driven by the rise in missed fraud. Under the targeted adversary, the opposite pattern emerged: recall increased and FNR decreased with compromise, as coordinated high-confidence FRAUD votes drove additional override activations at the cost of increased FPR and reduced precision; cost decreased monotonically, driven by the reduction in false negatives. Notably, results at 25% and 50% compromise are identical under the targeted adversary across all fraud rate conditions, reflecting the $k = 3$ supermajority threshold: fewer than 3 compromised agents cannot independently reach supermajority, leaving the override mechanism unaffected until 75% compromise.

Table 7 Performance Under Adversarial Conditions: Targeted

FR (%)	Compromise (%)	Precision	Accuracy	Recall	FPR	FNR	Cost
5%	0	64.7%	96.7%	76.0%	2.2%	24.0%	2607
	25	51.5%	95.3%	86.0%	4.2%	14.0%	1700
	50	51.5%	95.3%	86.0%	4.2%	14.0%	1700
	75	48.0%	94.6%	88.0%	5.1%	12.0%	1681
10%	0	79.4%	95.7%	76.5%	2.2%	23.5%	4899
	25	69.0%	94.7%	85.0%	4.1%	15.0%	3200
	50	69.0%	94.7%	85.0%	4.1%	15.0%	3200
	75	65.5%	94.2%	88.1%	5.1%	11.9%	2842
20%	0	88.9%	93.4%	74.9%	2.2%	25.1%	9779
	25	82.5%	93.6%	84.0%	4.1%	15.0%	6000
	50	82.5%	93.6%	84.0%	4.1%	15.0%	6000
	75	80.0%	93.4%	87.6%	5.2%	12.4%	5160

Accuracy remained above 93% under both adversary types across all conditions. Under the vote-flip adversary, accuracy counterintuitively increased with agent compromise at lower fraud prevalences (96.7% $\rightarrow$ 98.0% at 5% fraud rate) before marginal decline at 75%, reflecting anchor dominance: suppressed overrides revert the system toward the ML baseline, which achieves high accuracy by class composition at low fraud prevalence rather than improved fraud detection. This underscores why accuracy alone is insufficient to characterize adversarial robustness. The targeted adversary represents a more operationally realistic threat, exploiting the override mechanism in a manner the vote-flip adversary cannot. Agent confidence calibration under adversarial conditions was not assessed and represents a limitation of the adversarial evaluation.

Theorem Validation

For Theorem 1, Theorem 2, Lemma 1, and Theorem 4, properties that follow directly from the algorithm's definition, validation confirms that the implementation correctly realizes its specification; a violation would indicate an implementation bug rather than a failure of the underlying claim. For Corollary 1, Theorem 3, and Corollary 2, derived consequences of these definitional properties, validation provides empirical

Table 8 Theoretical Guarantees – Empirical Validation

Property	Theoretical Guarantee	Empirical Validation	Violations
Monotonicity (Theorem 1)	$\hat{y}_{ML} = 1 \Rightarrow \hat{y}_{final} = 1$	500 ML fraud verdicts: 100% preserved	0
Recall Lower Bound (Corollary 1)	$\text{Recall}_{final} \geq \text{Recall}_{ML}$	76.0% > 65.4%	0
Override Necessity (Theorem 2)	$o = 1 \Leftrightarrow$ 3 conditions met	53 overrides: all met 3 conditions	0
Confidence Respect (Lemma 1)	$p_{ML} \geq \tau \Rightarrow o = 0$	0 high-conf ML predictions overridden	0
Asymmetric Escalation (Theorem 3)	Can escalate; cannot de-escalate	53 SAFE → FRAUD; 0 FRAUD → SAFE	0
Conservative Bias (Corollary 2)	$FN_{final} \leq FN_{ML}$; $FP_{final} \geq FP_{ML}$	FN: 120 < 173; FP: 207 > 31	0
Decision Completeness (Theorem 4)	All transactions classified	10,000 classified in 3 regions	0
Computational Efficiency (Theorem 5)	$O(T)$ with parallelization	Parallel execution confirmed; runtime complexity not empirically benchmarked	0

confirmation of the corresponding behavioral guarantees (recall floor, escalation asymmetry, and conservative bias) across the evaluation dataset. Formal properties were verified at the 5% fraud rate condition ($n = 10{,}000$); results are reported in Table 8.

Discussion

The ML-Anchored Supermajority Consensus algorithm improved recall over the Random Forest baseline by approximately 10 percentage points across all three fraud rate conditions, with statistically significant results well below $\alpha = 0.05$ and large effect sizes (Cohen's $d = 6.335$–27.374). The increase in Cohen's d across fraud rate conditions reflects decreasing within-condition variance rather than a growing absolute treatment effect, as the mean recall difference remained stable across all three conditions (approximately 10 percentage points throughout). However, per-dataset analysis reveals that all recall gains are attributable to Bitcoin Heist; on Bitcoin Elliptic the supermajority override was never triggered, indicating the architecture's benefit is conditional on the ML baseline exhibiting systematic recall limitations.

The recall improvement came at a substantial precision cost across all fraud rate conditions. Under the asymmetric cost model (Cost = FP + 20×FN), the reduction in false negatives could result in lower aggregate cost, but the increase in false positive volume warrants consideration in deployment contexts where analyst capacity is constrained. The precision gap narrowed as fraud prevalence increased, partly due to the shift in class composition reducing the pool of legitimate transactions available to generate false positives, suggesting the precision cost may be more operationally acceptable in higher-prevalence environments.

The adversarial evaluation revealed a meaningful distinction between adversary types. The vote-flip adversary could not meaningfully exploit the architecture, as suppressed overrides pushed the system toward the ML baseline. The targeted adversary, coordinating high-confidence FRAUD votes on legitimate transactions, degraded precision substantially while improving recall, exposing the override mechanism's vulnerability to coordinated false positive inflation. Accuracy remained above 93% under both adversary types, reflecting the stabilizing influence of the ML anchor, though accuracy alone is insufficient to characterize robustness. These findings represent empirical robustness observations rather than formal Byzantine fault tolerance guarantees in the distributed systems sense.

Inter-agent agreement (Fleiss' $\kappa = 0.289$–0.407) indicated fair to moderate agreement that increased with fraud prevalence. The Consensus Builder exhibited near-perfect alignment with the ML baseline ($\kappa = 0.98$–0.99), while the Security Watchdog and Adversarial Analyst showed substantially lower baseline agreement, suggesting meaningful behavioral differentiation through prompt design. However, all four agents share the same underlying GPT-4o-mini model; correlated reasoning across agents sharing the same model weights represents a limitation that prompt differentiation cannot fully resolve.

The study has several limitations. The evaluation dataset of 10,000 transactions per fraud rate condition limits statistical power for rare fraud patterns. The confidence gating mechanism introduces a vulnerability wherein adversarially crafted inputs could trigger high-confidence erroneous baseline predictions, bypassing the override entirely. Agent confidence calibration under adversarial conditions was not assessed. LLM deployment introduces inference latency, API overhead, and scalability costs not captured by the theoretical $O(T)$ complexity bound; runtime benchmarking was not conducted. Production deployment would require self-hosted LLMs to maintain data privacy compliance. Generalizability to other blockchain ecosystems and fraud categories such as Ponzi schemes and pump-and-dump manipulation remains to be established. Future work should address architectural agent diversity through multiple model families, empirical runtime benchmarking, and evaluation across additional datasets and fraud categories.

References

1 M. Weber, G. Domeniconi, J. Chen, D. K. I. Weidele, C. Bellei, T. Robinson and C. E. Leiserson, *Anti-money laundering in Bitcoin: experimenting with graph convolutional networks for financial forensics*, 2019, 10.48550/arXiv.1908.02591.

2 C. G. Akcora, Y. Li, Y. R. Gel and M. Kantarcioglu, *BitcoinHeist: Topological data analysis for ransomware prediction on the Bitcoin blockchain*, 2020, 10.24963/ijcai.2020/612.

3 T. Ashfaq, R. Khalid, A. Yahaya, S. Aslam, A. K. Nauman, O. Alfandi and F. Al-Obeidat, *A machine learning and blockchain based efficient fraud detection mechanism*, 2022, 10.3390/s22197162.

4 A. Venčkauskas, Grigaliūnas, L. Pocius, R. Brūzgienė and A. Romanovs, *Machine learning in money laundering detection over blockchain technology*, 2024, 10.1109/ACCESS.2024.3452003.

5 I. Alarab, S. Prakoonwit and M. I. Nacer, *Comparative analysis using supervised learning methods for anti-money laundering in Bitcoin*, 2020, 10.1145/3409073.3409078.

6 L. Breiman, *Random forests*, 2001, 10.1023/A:1010933404324.

7 Y. Freund and R. E. Schapire, *A decision-theoretic generalization of online learning and an application to boosting*, 1997, 10.1006/jcss.1997.1504.

8 T. G. Dietterich, *Ensemble methods in machine learning*, 2000, 10.1007/3-540-45014-9_1.

9 D. Opitz and R. Maclin, *Popular ensemble methods: an empirical study*, 1999, 10.1613/jair.614.

10 Z. Gu and O. Dib, *Enhancing fraud detection in the Ethereum blockchain using ensemble learning*, 2025, 10.7717/peerj-cs.2716.

11 G. Pahuja and T. N. Kamal, *ENLEFD-DM: Ensemble learning based Ethereum fraud detection using CRISP-DM framework*, 2023, 10.1111/exsy.13379.

12 M. A. Quadir, M. A. Yousuf, M. S. Uddin, M. S. Kaiser and M. A. Islam, *A novel approach to detect fraud in Ethereum transactions using stacking*, 2023, 10.1111/exsy.13255.

13 M. K. Taher, B. A. Ameen and H. S. Ahmed, *Advanced fraud detection in blockchain transactions: an ensemble learning and explainable AI approach*, 2024, 10.48084/etasr.6641.

14 L. Zhang, Y. Xuan, Z. Liu, Z. Du, S. Wang and J. Wang, *A hybrid ensemble model to detect Bitcoin fraudulent transactions*, 2025, 10.1016/j.engappai.2024.109810.

15 S. Kim, Y. Kim, J. Lee and J. Shim, *Graph learning-based blockchain phishing account detection with a heterogeneous transaction graph*, 2023, 10.3390/s23010463.

16 W. W. Lo, G. K. Kulatilleke, M. Sarhan, S. Layeghy and M. Portmann, *Inspection-L: self-supervised GNN node embeddings for money laundering detection in Bitcoin*, 2023, 10.1007/s10489-023-04504-9.

17 L. Lamport, R. Shostak and M. Pease, *The Byzantine generals problem*, 1982, 10.1145/357172.357176.

18 M. Castro and B. Liskov, *Practical Byzantine fault tolerance*, 1999, https://www.usenix.org/conference/osdi-99/practical-byzantine-fault-tolerance.

19 R. Olfati-Saber, J. A. Fax and R. M. Murray, *Consensus and cooperation in networked multi-agent systems*, 2007, 10.1109/JPROC.2006.887293.

20 W. Ren and R. W. Beard, *Distributed Consensus in Multi-vehicle Cooperative Control*, 2008, 10.1007/978-1-84800-015-5.

21 M. H. DeGroot, *Reaching a consensus*, 1974, 10.1080/01621459.1974.10480137.

22 L. Zheng, J. Chen, Q. Yin, J. Zhang, X. Zeng and Y. Tian, *Rethinking the reliability of multi-agent system: a perspective from Byzantine fault tolerance*, 2026, 10.1609/aaai.v40i41.40806.

23 Z. Chen, Y. Li, J. Lin, Z. Wang and J. Li, *BlockAgents: towards Byzantine-robust LLM-based multi-agent coordination via blockchain*, 2024, 10.1145/3674399.3674445.

24 J. Huang, J. Zhou, T. Jin, X. Zhou, Z. Chen, W. Wang, Y. Yuan, M. Lyu and M. Sap, *On the resilience of LLM-based multi-agent collaboration with faulty agents*, 2025, https://proceedings.mlr.press/v267/huang25ay.html.

25 T. Liang, Z. He, W. Jiao, X. Wang, Y. Wang, R. Yang, Y. Shi and S. Z. Tu, *Encouraging divergent thinking in large language models through multi-agent debate*, 2024, 10.18653/v1/2024.emnlp-main.992.

26 J. Chen, S. Saha and M. Bansal, *ReConcile: round-table conference improves reasoning via consensus among diverse LLMs*, 2024, https://aclanthology.org/2024.acl-long.382.

Appendix

Theoretical Guarantees – Formal Proofs

Theorem 1 (Monotonicity Guarantee)

Formal Statement:

$$\forall x \in \mathbb{R}^d,\ \hat{y}_{ML}(x) = 1 \Rightarrow \hat{y}_{final}(x) = 1 \tag{3}$$

Proof: Since $\hat{y}_{final} = \hat{y}_{ML} \vee o$ and $o \in \{0,1\}$, we have $1 \vee o = 1$ for either value of o. Therefore $\hat{y}_{ML} = 1$ implies $\hat{y}_{final} = 1$ regardless of the override indicator. Fraud verdicts cannot be dismissed.

Corollary 1 (Recall Lower Bound)

Corollary Statement: The algorithm's recall cannot fall below the ML baseline recall.

Formal Statement:

$$\text{Recall}_{final} \geq \text{Recall}_{ML} \tag{4}$$

Proof: By Theorem 1, $TP_{ML} \subseteq TP_{final}$, since every transaction the ML baseline flags as fraud is also flagged by the algorithm. The override mechanism (Theorem 2) can additionally escalate SAFE-labeled transactions to FRAUD when agents reach supermajority, so $TP_{final} \geq TP_{ML}$ and correspondingly $FN_{final} \leq FN_{ML}$. Since Recall $= TP/(TP+FN)$ is non-decreasing in TP for fixed total positives, $\text{Recall}_{final} \geq \text{Recall}_{ML}$. The ML anchor thus acts as a performance floor, with agent consensus providing only upward corrections.

Theorem 2 (Override Necessity)

Theorem Statement: The override mechanism activates if and only if three conditions are simultaneously satisfied: (1) ML baseline predicts SAFE, (2) supermajority of agents predict FRAUD, and (3) ML baseline exhibits low confidence.

Formal Statement:

$$o = 1 \Leftrightarrow (\hat{y}_{ML} = 0) \wedge (s \geq k) \wedge (p_{ML} < \tau) \tag{5}$$

where s represents the sum of agent votes for fraud, k is the supermajority threshold, and τ is the confidence threshold.

Proof: This is immediate from the definition $o = \mathbb{1}[\hat{y}_{ML} = 0 \wedge s \geq k \wedge p_{ML} < \tau]$, where $\mathbb{1}[\cdot]$ is the indicator function: $o = 1$ exactly when its argument holds, and $o = 0$ otherwise.

Lemma 1 (Confidence-Conditional Override)

Statement: If the ML baseline predicts SAFE with high confidence ($p_{ML} \geq \tau$), override cannot activate regardless of agent votes.

Proof: By Theorem 2, $o = 1$ requires $p_{ML} < \tau$. If $p_{ML} \geq \tau$, this condition fails, so $o = 0$ irrespective of s.

Theorem 3 (Asymmetric Escalation)

Theorem Statement: The algorithm can escalate verdicts from SAFE to FRAUD via override but cannot de-escalate verdicts from FRAUD to SAFE.

Formal Statement:

(a) Escalation Capability:

$$\exists x \in \mathbb{R}^d : \hat{y}_{ML}(x) = 0 \wedge \hat{y}_{final}(x) = 1 \tag{6}$$

(b) De-escalation Impossibility:

$$\nexists x \in \mathbb{R}^d : \hat{y}_{ML}(x) = 1 \wedge \hat{y}_{final}(x) = 0 \tag{7}$$

Proof: (a) For any x with $\hat{y}_{ML} = 0$, $p_{ML} < \tau$, and $s \geq k$, Theorem 2 gives $o = 1$, so $\hat{y}_{final} = 0 \vee 1 = 1$, demonstrating escalation. (b) By Theorem 1, $\hat{y}_{ML} = 1 \Rightarrow \hat{y}_{final} = 1$; equivalently (by contraposition), $\hat{y}_{final} = 0 \Rightarrow \hat{y}_{ML} = 0$. Hence $\hat{y}_{ML} = 1 \wedge \hat{y}_{final} = 0$ cannot occur.

Corollary 2 (Conservative Bias)

The algorithm exhibits systematic conservative bias favoring false positives over false negatives.

Proof Sketch: By Theorem 3(a), escalation can convert ML false negatives into true positives, so $FN_{final} \leq FN_{ML}$. By Theorem 3(b), no FRAUD verdict is ever de-escalated, so escalation onto SAFE-labeled negatives can only increase false positives: $FP_{final} \geq FP_{ML}$. The algorithm thus trades increased FP for decreased FN.

Theorem 4 (Decision Regions)

The algorithm partitions the input space $\mathbb{R}^d$ into three disjoint decision regions.

Region 1 (Monotonic Fraud): $R_1 = \{x \in \mathbb{R}^d : \hat{y}_{ML}(x) = 1\}$

Region 2 (Override-Escalated Fraud): $R_2 = \{x \in \mathbb{R}^d : \hat{y}_{ML}(x) = 0 \wedge s(x) \geq k \wedge p_{ML}(x) < \tau\}$

Region 3 (Approved Safe): $R_3 = \{x \in \mathbb{R}^d : \hat{y}_{ML}(x) = 0 \wedge (s(x) < k \vee p_{ML}(x) \geq \tau)\}$

Proof: Disjointness follows since R_1 requires $\hat{y}_{ML} = 1$ while R_2, R_3 require $\hat{y}_{ML} = 0$. Exhaustiveness follows by case analysis on $\hat{y}_{ML}(x)$: if $\hat{y}_{ML}(x) = 1$, $x \in R_1$; if $\hat{y}_{ML}(x) = 0$, either $(s \geq k \wedge p_{ML} < \tau)$ holds, placing $x \in R_2$, or its negation holds (by De Morgan's law), placing $x \in R_3$. Final predictions follow directly: $\hat{y}_{final} = 1$ on R_1 (Theorem 1) and on R_2 (Theorem 2 gives $o = 1$, so $\hat{y}_{final} = \hat{y}_{ML} \vee o = 0 \vee 1 = 1$); $\hat{y}_{final} = 0$ on R_3 ($o = 0$, so $\hat{y}_{final} = 0 \vee 0 = 0$).