On-Chain Decentralized Learning and Cost-Effective Inference for DeFi Attack Mitigation

Abdulrahman Alhaidari
School of Computing and Information, University of Pittsburgh, Pittsburgh, PA, USA

Balaji Palanisamy
School of Computing and Information, University of Pittsburgh, Pittsburgh, PA, USA

Prashant Krishnamurthy
School of Computing and Information, University of Pittsburgh, Pittsburgh, PA, USA

Abstract

Billions of dollars are lost every year in DeFi platforms by transactions exploiting business logic or accounting vulnerabilities. Existing defenses focus on static code analysis, public mempool screening, attacker contract detection, or trusted off-chain monitors, none of which prevents exploits submitted through private relays or malicious contracts that execute within the same block. We present the first decentralized, fully on-chain learning framework that: (i) performs gas-prohibitive computation on Layer-2 to reduce cost, (ii) propagates verified model updates to Layer-1, and (iii) enables gas-bounded, low-latency inference inside smart contracts. A novel Proof-of-Improvement (PoIm) protocol governs the training process and verifies each decentralized micro update as a self-verifying training transaction. Updates are accepted by *PoIm* only if they demonstrably improve at least one core metric (e.g., accuracy, F1-score, precision, or recall) on a public benchmark without degrading any of the other core metrics, while adversarial proposals get financially penalized through an adaptable test set for evolving threats. We develop quantization and loop-unrolling techniques that enable inference for logistic regression, SVM, MLPs, CNNs, and gated RNNs (with support for formally verified decision tree inference) within the Ethereum block gas limit, while remaining bit-exact to their off-chain counterparts, formally proven in Z3. We curate 298 unique real-world exploits (2020 - 2025) with 402 exploit transactions across eight EVM chains, collectively responsible for \$3.74 B in losses. We demonstrate that on-chain ML governed by *PoIm* detects previously unseen attacks with over 97% attack detection accuracy and 82.0% F1. A single inference, such as one made via an external call, typically incurs zero cost. Fully on-chain inference consumes 57,603 gas ($\approx$ \$0.18) for linear models, 143,647 gas ($\approx$ \$0.49) for CNN(F2, K1), and 506,397 gas ($\approx$ \$1.77) for CNN(F8, K4) on L1 (e.g., Ethereum). Our results show that practical and continually evolving DeFi defenses can be embedded directly in protocol logic without trusted guardians, and our solution achieves highly cost-effective protection while filling a critical gap between vulnerability scanners and real-time transaction screening.

2012 ACM Subject Classification Security and privacy → Network security; Security and privacy → Distributed systems security; Security and privacy → Security protocols

Keywords and phrases DeFi attacks, on-chain machine learning, decentralized learning, real-time defense

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.35

Funding This material is based upon work supported by the National Science Foundation under Grant #2020071. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

7th Conference on Advances in Financial Technologies (AFT 2025).
Editors: Zeta Avarikioti and Nicolas Christin; Article No. 35; pp. 35:1–35:27
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Vulnerabilities in decentralized finance (DeFi) protocols are triggered through transactions [8]. Attackers often do not bypass contract safeguards directly [55], instead they craft transactions that invoke legitimate functions to trigger state changes that the protocol did not intend. Exploits occur when contracts proceed with malicious inputs while assuming invariant state conditions, such as executing a withdrawal without checking balance or allowance or privileged operations without enforcing access control [50]. These attacks can be single transactions or atomic sequences that satisfy syntax checks but produce unauthorized asset control. The execution logic can be formally valid, yet produce outcomes that violate the protocol's security assumptions [9]. Vulnerabilities come from code bugs and manipulating contracts. Attackers often chain operations (for example, in atomic transactions) that appear normal in unexpected ways to generate exploits [12]. However, most previous work focuses on static code-level bugs [52] and overlooks protocol-level flaws [68].

Static analysis detects patterns that violate predefined coding conventions, but fails to capture logic flaws that depend on contract state, cross-function flows, or interactions across multiple protocols [10]. For instance, Cream Finance lost over $130M through an interaction with another protocol that allowed borrowing without triggering the appropriate collateral checks [47]. Formal verification tools prove that certain invariants hold under all code paths but do not encode financial semantics or simulate attacker incentives. Most verification frameworks cannot model adversarially composed transaction sequences, in which attackers combine individually valid operations to produce exploits. Thus, zero-day vulnerabilities continue to appear in DeFi [68]. Tools for detecting malicious smart contracts assume a time window between contract deployment and first exploitation [54], during which vulnerabilities can be analyzed. In practice, attackers often deploy and execute the exploit within a single block or bypass deployment entirely by sending malicious transactions from externally owned accounts (EOA) [41].

Other methods, such as post-attack analysis, provide insight into what has happened but offer no protection. They begin only after an exploit has occurred and rely on retrospective debugging of protocol states [19]. However, recent attacks in DeFi have compromised over $79.8 billion in DeFi assets and only $6.7 billion of them have been recovered [18]. This emphasizes a core challenge in shared threat intelligence, particularly for aftermath attack analysis, where dissemination is often delayed. This delay increases the likelihood of repeat attacks on existing deployed protocols that may harbor the same unaddressed vulnerabilities. Methods such as front-running protection [67] monitor public mempools (public queue of transactions) but miss transactions submitted through private relays such as Flashbots [25]. While off-chain monitoring systems (e.g., [67, 6, 54]) can detect some attacks, they typically focus on specific classes (e.g., flash loans, reentrancy). Moreover, their effectiveness is diminished by latency, which can lead to costly responses (due to gas fees) and reduced efficacy, particularly against sophisticated attacks [61]. Therefore, there is a need for protocol-integrated security mechanisms that add an evolving layer of protection (i.e., Intrusion Prevention System (IPS)) to smart contracts and react in real time without relying on external entities, such as pre-attack (e.g., auditing) or post-attack (i.e., attack tracing) countermeasures. On the other hand, DeFi protocols currently lack mechanisms to evaluate transaction intent during execution other than hard-coded logic, which makes them vulnerable to attacks that were not anticipated during smart contract design.

Solutions driven by machine learning (ML) show strong capabilities in attack detection [33, 37]. However, deploying ML models directly to Layer-1 blockchains such as Ethereum faces major obstacles in computation and storage. The computational demands of ML algorithms result in prohibitive gas costs. Ethereum's block gas limits ($\approx$ 30 million units per block [26]) impose constraints on the computational complexity of transactions, making complex ML models impossible to run in a single transaction. A basic model inference might consume 30-50% of an entire block. Storage costs create another barrier. Saving model parameters on-chain is extremely expensive [37]. These combined limitations make direct Layer-1 deployment of ML solutions economically and technically impractical for real-world deployment.

To address the above-mentioned gaps, we propose a decentralized training architecture where all model training and governance occur on Layer-2 (L2), while inference is optimized and happens on Layer-1 (L1) under strict gas constraints (Table 1). In our framework, an ML model is trained to provide a layer of security for smart contracts running on the Ethereum (L1) network. Our framework leverages L2 (e.g., Optimism rollup) for intensive computation and L1 for optimized inference to overcome L1 resource constraints (such as computation limits and gas fees). The rollup provides cost-effective computation while inheriting the base blockchain's security guarantees [60]. Training is performed on L2, and L1 updates are governed by L2 decentralized nodes, where computation is cheaper, and the learned model is cryptographically verified and propagated to L1 for inference. Inference runs at zero cost through read-only classifiers (for example, pure or view functions) or is fully verified on L1 embedded in on-chain contracts for real-time transaction classification. To support a range of use cases, we propose two tiers: *fully zero-cost inference* (for users and smart contracts) on L1, and *fully on-chain ML on L1*. The system acts as a transaction gatekeeper (i.e., firewall) even for high-throughput or low-value use cases. Our approach is model-agnostic and supports various ML algorithms, including linear algorithms (e.g., logistic regression, SVMs) and non-linear models (e.g., neural networks up to 10 layers, including 10-layer convolutional neural networks (CNN)). These models are optimized and serialized into constant-time evaluation logic fully bounded by L1 constraints. We found that even low-overhead mechanisms (linear models) are sufficient to detect a wide range of attacks. The models are trained using micro-steps by decentralized peers based on the collective knowledge of the DeFi platforms, and their formally verifiable performance matches traditional on-chain counterparts without approximation, enabling detection of both known and novel transaction behaviors.

As DeFi exploits evolve over time and to maintain continuous learning of new exploits as they appear, we introduce the notion of *Proof-of-Improvement (PoIm)*. *PoIm* is a decentralized protocol on L2 that governs and verifies micro-step model training and its L1-propagated updates. A deployed model is designed to be shared globally across DeFi protocols, allowing any platform to contribute by training it on one candidate (malicious or normal) transaction at a time. These DeFi platforms collectively shape a model that becomes robust over time and enables unified sharing of attack intelligence. All updates are evaluated against an on-chain (L2) committed, agreed-upon benchmark of past exploit and benign transaction data. Updates are accepted only if they demonstrably improve at least one of the key performance metrics (namely precision, recall, accuracy, or F1-score) without degrading others, as evaluated by *PoIm* against the on-chain benchmark and it is verifiable by any node. Submitters are rewarded in proportion to the verified performance gain, while failed proposals lose their stake. If a malicious update seemingly improves metrics but enables a detection bypass, peers can vote to roll back the model to the last stable version. *PoIm* enables the L1 classifier to evolve with newly emerging exploits without relying on centralized oversight.

Table 1 Comparison of our work with prior on-chain ML studies. **DT** = Decentralized Training, **MC** = Model Consistency (On-/Off-chain), **G** = Governance, **DF** = DeFi Focus, **IC** = Inference Cost, **TM** = Trust Model, **VM** = Validation Method.

Study	DT	MC	G	DF	IC	TM	VM
Our Work	✓	✓	✓	✓	Zero / Low	*PoIm*	On-chain + commit-reveal
ML2SC [42]	✗	*Partial*[1]	✗	✗	High	✗	✗
LMST [57]	✗	*Partial*[2]	✗	✗	Mod-High	✗	✗
opML [15]	✗[3]	✓	✓	✗	Low*	AnyTrust	Fraud-proof

[1] Minor mismatches due to PRBMath; [2] Accuracy drop from fixed-point; [3] Training off-chain, validated via fraud proof; * Optimistic assumption, only $O(1)$ on-chain arbitration.

For our evaluation, since there is no publicly available transaction data for DeFi attacks, we manually collected 298 confirmed exploit transactions from real DeFi attacks across eight major blockchains: *Ethereum, Binance Smart Chain (BSC), Polygon, Avalanche, Arbitrum, Fantom, Moonriver, and Base.* Our exploit collection generalizes to cover attacks that exploited smart contract vulnerabilities in the past five years (2020 - April 2025), reported in news, social media platforms (e.g., X), blogs, and DeFi attack documentation. We utilize blockchain data from public explorers (for example, Etherscan [22]) and Web3 Remote Procedure Calls (RPCs) (e.g., using libraries/services like Web3.js [14] and Alchemy [4]), along with DeFiHackLabs [58], the DeFi Rekt Database [18], and DeFiLlama [19] as ground truth for guiding historical exploit collection. Each transaction contains the exact exploit transaction data used by the attacker, as observed by the smart contract at execution time, including call parameters, sender addresses (EOAs or smart contracts), and relevant blockchain state at the time of the exploit. In addition, we collected comprehensive metadata for each attack, including protocol names, exploited functions, attack methods, root causes, and financial losses. We use these real-world exploits, which collectively caused over $3.74 billion in losses. Our evaluation shows that our approach is highly gas-efficient. CNN and RNN models governed by our decentralized *PoIm* protocol achieve over 97% recall and up to an 82% F1-score on unseen DeFi exploits.

Contributions. This paper makes the following contributions:

- We design a decentralized learning framework with training and governance on L2 and enable two tiers: zero-cost inference and fee-optimized inference on L1 for real-time classification of DeFi transactions.
- We introduce *PoIm*, a decentralized L2 protocol that governs model training and propagates verified updates to L1 for inference.
- We formally verify inference correctness, model update integrity, and L1/L2 consistency under gas and computation constraints.
- We evaluated our framework on a curated set of 298 manually collected real-world exploits. It achieves high detection performance: SVM reaches an F1-score of 80%, and CNN(F4, K4) achieves 82% F1, 90.04% accuracy, and over 97% recall on unseen exploits. L1 inference is efficient, requiring only 57,603 gas for linear models and 143,647 gas for CNN(F2, K1). Zero-cost inference is supported via external EVM nodes.

2 Background

Layer 2 Blockchains. Layer 2 (L2) blockchains (e.g., Optimism, Arbitrum, Base) are built on top of Layer 1 (L1) platforms (e.g., Ethereum, Binance Smart Chain (BSC)) to improve scalability and reduce costs [56]. In contrast to L1s that execute and store every transaction directly on-chain [26], L2s batch many transactions and periodically submit compressed proofs to the L1 [63]. This design allows L2s to offer significantly lower gas fees and faster execution and inherits the security guarantees from L1 [56]. Notably, L2s support the same smart contract logic as L1s but with relaxed resource constraints, which makes them ideal for computationally heavy tasks like model training or repeated inference.

Transactions. Transactions are the main component used to facilitate both asset movement and protocol interactions. A nonce in a transaction acts as a sequence counter to prevent replay attacks, where the attacker could replicate the transaction. The gas fee, influenced by user-specified price and the complexity of transactions, determines inclusion priority, while the gas limit defines the upper computational allowance per transaction [62].

Exploit transactions. Most of the DeFi attacks succeed through a maliciously crafted sequence of instructions (e.g., internal transactions) [20]. These attacks exploit weaknesses in smart contracts rather than modifying contract code at the infrastructure level [71]. Attackers manipulate transaction parameters, call sequences, and attempt to manipulate permission states through public interfaces to gain unauthorized assets [55].

Mempool and private relays. Blockchain transactions are submitted either through the public transaction buffer, known as the mempool, or via private relays [25], where the transaction is sent directly to miners. Transactions submitted via private relays remain hidden from the public, whereas those using the mempool are broadcast and await confirmation [13]. Nodes share this queue across the network. The selection of which transactions to include in a block is typically based on miners/validators' incentives and the gas fees offered. Transactions with higher fees are generally prioritized [43]. The mempool is publicly visible, enabling a brief window during which real-time monitoring can be used to detect malicious behavior, such as front-running [36]. Since each node maintains a synchronized copy of unconfirmed transactions, this window supports early threat detection [31]. However, detection is not always possible due to the complexity of some attacks, and in many cases, the malicious transaction is submitted directly to miners/validators, bypassing public visibility [46].

2.1 Limitations of Existing Defenses

This section discusses fundamental limitations of existing preventive measures and real-time defense mechanisms. First, off-chain detection systems such as LookAhead [54] rely on a temporal gap between transaction submission and execution. When an attacker deploys a malicious contract and immediately initiates an exploit within a single block, these systems fail to respond on time. Second, private relay services like Flashbots [25] allow attackers to bypass mempool-based detection by submitting transactions directly to miners/validators. In such settings, traditional monitoring tools lose all visibility, allowing stealthy attacks without external traceability.

Third, signature-based detection methods depend on identifying known function selectors or matching call patterns against static signatures [6]. This approach fails when attackers use proxy contracts, delegate calls, or obfuscated logic flows, where surface-level transaction signatures are intentionally masked. Fourth, current on-chain defenses lack the ability to validate transaction behavior dynamically during execution. Existing systems either rely on static pre-deployment audits or basic access control checks, leaving dynamic runtime behavior, such as unauthorized fund movements or privilege escalations, undetected.

Fifth, most prior defenses prioritize contract vulnerabilities rather than transaction behavior. However, even with this focus, a recent study [10] found that automated tools can detect only 8% of vulnerabilities. Another study found that existing tools can detect only 20% of vulnerabilities [68], which leaves 80% undetected. Consequently, they can be exploited and cause huge financial losses. One reason is that not all bugs are exploitable, and not all correct logic code is vulnerability-free [50]. Thus, the focus on code logic overlooks the fact that many exploits occur through crafted transaction sequences exploiting valid contract interfaces without exploiting traditional code vulnerabilities.

Also, existing solutions typically adopt a centralized approach. Systems that depend on trusted off-chain detectors or external alert mechanisms inherently introduce single points of failure and trust dependencies incompatible with DeFi principles. Pre-interaction analysis tools such as DeFiAligner [28] can detect inconsistencies between a project's documentation and on-chain bytecode. However, they do not provide real-time defense against malicious transaction behavior.

2.2 Limitations of On-Chain ML

Embedding machine learning models into L1 smart contracts is resource-intensive, and replicating off-chain models directly on-chain can be impossible. L1 blockchains are not designed for heavy computing [3]. However, inference, if designed without inheriting the complexity of off-chain models (e.g., using TensorFlow [2], Keras [38], Scikit-learn [39]) yet performing equivalently, can be feasible if heavy computations are separated from inference.

Previous designs (e.g., [42, 57]) face many practical barriers, including prohibitive gas costs for inference, inconsistent behavior between off-chain and on-chain models due to numerical limitations, and incompatibility with smart contract languages lacking native support for floating-point operations, which causes model output deviation (off-chain vs on-chain counterpart). For example, the authors in [57] reported inconsistent accuracy between the PyTorch and on-chain models: 86.00% off-chain versus 81.00% on-chain. The deployment cost also exceeds the current Ethereum block gas limit, with a reported cost of 73,721,648 gas. This is more than twice Ethereum's current block gas limit, which is approximately 30 million gas units.

Our work achieves exact consistency (formally and empirically verified) between off-chain and on-chain model outputs. Unlike prior methods, our on-chain design uses an optimized smart contract that replicates the off-chain model's behavior without approximation to achieve the same outcome. This will allow DeFi attacks mitigation dynamically using practical, low-cost, and fully decentralized on-chain machine learning.

3 DeFi Attacks and Threat Model

The current state of defenses is not sufficient to mitigate the growing threat of unauthorized fund exploits in Decentralized Finance (DeFi). Recent studies [10, 40, 13, 71, 68] show that attack techniques are rapidly advancing but defenses remain inadequate. To address these threats, we need a new defense model based on a real-time, in-protocol (smart contract-level) transaction classification system that must operate independently of external monitors or pre-defined signature lists [71], and it must work even when transactions are hidden from the public mempool. It should also generalize to both known and novel (zero-day) attacks that exploit DeFi contract flaws. However, statistical methods have shown significant promise in detecting not only known attacks but also previously unseen zero-day exploits [33]. This naturally raises several key questions: Can data from past DeFi attacks help us predict and

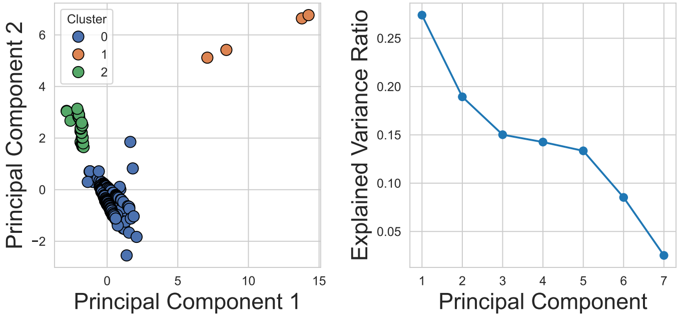

Figure 1 Left: PCA projection of DeFi attack transactions using two principal components with KMeans clustering ($k = 3$). Right: The scree plot shows the explained variance ratio per principal component. Even though PC1 and PC2 capture less than 50% of the total variance, the projection shows distinct DeFi attack transaction patterns.

prevent future ones? What are the possible designs for trustless, cost-effective, on-chain inference systems that can evolve with the attack landscape and be governed by decentralized peers? To answer these questions, we investigate whether real historical exploits exhibit detectable patterns that statistical techniques can reliably uncover and learn from.

3.1 Empirical Patterns of DeFi Attacks

To explore the feasibility of detecting DeFi attacks using only transaction data observable by smart contracts, we analyzed 402 confirmed DeFi exploit transactions (for 298 attacks). Transaction data includes complete EVM-level execution context such as gas usage, transferred value, calldata, and block metadata. Since previous studies offer limited analysis of DeFi attacks from a transactional perspective, we investigate whether exploit transactions exhibit measurable similarities that support transaction-level detection.

Table 2 Transaction metadata visible to smart contracts during execution.

Attribute	**Description**
`msg.sender`	Sender address (initiator of the call)
`msg.value`	ETH amount sent with the call
`msg.data`	Calldata containing function selector and arguments
`tx.origin`	Original externally-owned sender of the transaction
`gas`	Remaining gas (via `gasleft()`)
`gas_price`	Effective gas price paid (via `tx.gasprice`)
`msg.to`	Receiving contract’s address (via `address(this)`)

We selected the numerical features observable during smart contract execution (Table 2) and excluded transaction hashes or addresses. Features were standardized, and dimensionality was reduced using Principal Component Analysis (PCA). We applied KMeans clustering with $k = 3$ to detect grouping behavior across attack transactions.

The PCA projection preserved 46.3% of the variance in two dimensions (Figure 1). Out of 402 exploits, 355 (88.3%) formed a distinct cluster. This suggests high behavioral consistency across transactions despite protocol and chain differences. Clustering quality was supported by a Calinski-Harabasz index of 981.78 and a Silhouette score of 0.773, both indicating well-separated, compact clusters.

These findings indicate that DeFi attacks share consistent runtime characteristics. The ability of PCA + KMeans to cluster these attacks supports the feasibility of transaction-level classification during execution. We classify the root causes of these attacks into five empirically supported categories, such as access control failures and business logic flaws.

Exploit Execution Patterns. Despite different root causes, several runtime-level patterns are consistent across attacks:

- **Atomicity:** 71.2% of exploits execute within a single transaction with no prior on-chain activity.
- **Benign interface misuse:** 67.5% invoke functions like `withdraw()`, `approve()`, or `sweep()` with malicious arguments.
- **Cross-chain replication:** Identical bytecode deployments (9.7%) are exploited across chains [35] (e.g., Hedgey [1]).

These patterns further support the view that exploitability is determined by transaction behavior rather than only static code.

3.2 Challenges in Real-Time Defense

We identified several key challenges that prevent current systems from defending against these attacks in real time:

C1. **Private relay invisibility:** Transactions sent through Flashbots [25] and similar relays bypass the public mempool, evading pre-inclusion detection.

C2. **Multiple attack paths per contract:** A single protocol may contain unrelated flaws, making static patching or signature detection ineffective.

C3. **Benign-looking interfaces:** Safe-looking functions are abused with crafted inputs, undermining static signature-based detection.

C4. **Cross-chain propagation:** An undetected exploit on one chain quickly propagates to other protocols with shared logic.

C5. **No rollback:** Once executed, DeFi transactions are final. Most protocols do not have pause switches or delayed execution.

3.3 Threat Model and Assumptions

Our system defends against two primary threat scenarios. ① An attacker crafts a malicious transaction targeting a vulnerable contract. The transaction is evaluated by our on-chain ML classifier before execution. The attacker may vary calldata, timing, or submission channel (e.g., private relay), but cannot alter the deployed model or governance system. ② A malicious peer attempts to poison the training process by submitting manipulated updates that degrade model performance. The system detects and rejects such updates via on-chain benchmarking. Insider threats, compromised signing keys, and off-chain infrastructure attacks are out of scope for this threat model.

4 Decentralized Training, Inference, and Governance Framework

Our objective is to integrate decentralized learning with verifiable and efficient on-chain detection into the execution paths of DeFi contracts to mitigate exploit transactions in real time. We address limitations in existing works (e.g., [42, 67, 5, 54, 57]) from two perspectives: DeFi attack mitigation and on-chain ML design. First, we detect private relay attack transactions and generalize the detection mechanism to unseen attacks in real time. Second,

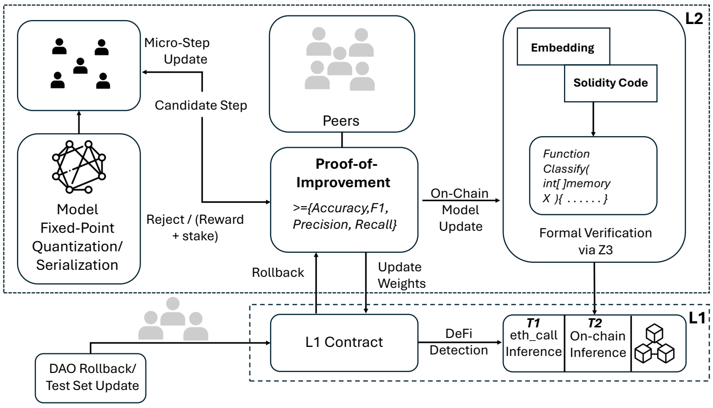
L2
Micro-Step Update
Candidate Step
Peers
Embedding
Solidity Code
Model Fixed-Point Quantization/ Serialization
Reject / (Reward + stake)
Proof-of-Improvement
>={Accuracy,F1, Precision, Recall}
On-Chain Model Update
Function Classify(int[ ]memory X){ . . . . . }
Formal Verification via Z3
Rollback
Update Weights
L1
L1 Contract
DeFi Detection
T1 eth_call Inference
T2 On-chain Inference
DAO Rollback/ Test Set Update

Figure 2 Framework overview.

we ensure bitwise consistency between off-chain and on-chain models, and cost-effective decentralized training and inference. We also guarantee bounded resource consumption, such as gas fees for training and inference. Furthermore, we introduce a decentralized mechanism that enables peers (i.e., DeFi platforms) to collectively train ML/DL models (e.g., for attack defense) by proposing training samples. These proposed updates are transparently governed, validated, and challengeable by others. An overview of our end-to-end framework is presented in Figure 2.

Our framework enables decentralized learning by translating traditional off-chain machine learning and deep learning model architectures into gas-efficient, formally verified Solidity contracts for blockchain execution via L2 rollups (e.g., Optimism [48]). It supports a wide range of architectures, from simple to complex, and produces tamper-proof ML/DL contracts with quantized parameters serialized for on-chain weight propagation (Figure 4). This capability not only enhances DeFi threat detection but also provides a foundation for developing adaptive and secure financial systems in other domains. Our approach addresses the limitations of prior work discussed in Section 2.2. Previous efforts (e.g., [42, 57]) are impractical and incur high gas costs when deploying complex models on L1, which can be very complex.

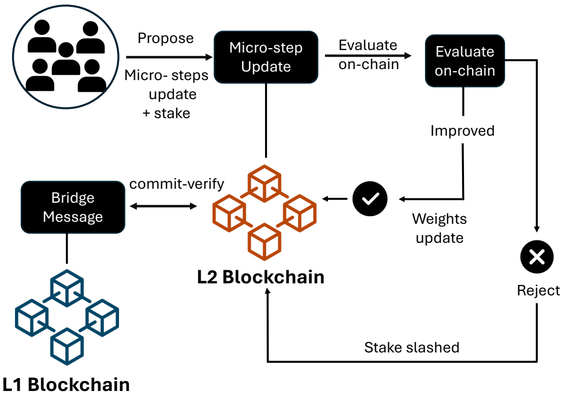

Figure 3 *PoIm* overview.

In contrast, our design supports full-scale, real-world models without compromising accuracy due to Solidity's floating-point limitations, and it enables fully decentralized training and model updates. To achieve this, we introduce Proof-of-Improvement (PoIm), a decentralized governance protocol that tracks each training step and governs model updates, as shown in Figure 3. It allows anyone (e.g., any DeFi platform) to incrementally train the model, effectively proposing updates that improve detection metrics. Updates are predictably propagated from L2, where computation and verification are performed, to L1, where inference takes place. Peers can collectively override a propagated model update on L1 in cases of suspected malicious training (poisoning) by another peer. Overall, our decentralized design is motivated by DeFi security requirements, and we show that a decentralized, dynamic, and cost-effective ML/DL-based defense is feasible under current blockchain architectures enabled by L2, capable of mitigating attacks that have caused billions in financial losses.

4.1 Decentralized Micro-Step Training and Model Evolution

Our system enables verifiable, decentralized training by decomposing the learning process into micro-step updates. Each micro-step corresponds to a single incremental improvement, such as retraining on one example or applying a localized adjustment, and is proposed directly on-chain. All proposals are immediately evaluated using a canonical, public test set stored on-chain.

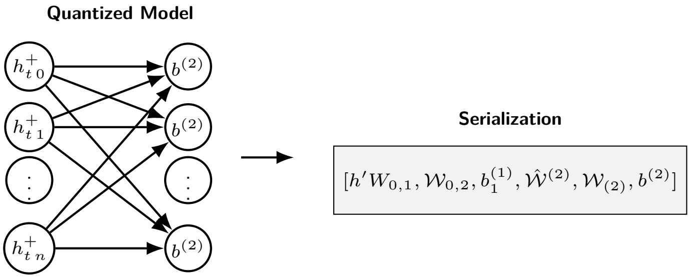
Quantized Model
$h^{+}_{t\,0}$
$h^{+}_{t\,1}$
$h^{+}_{t\,n}$
$b^{(2)}$
$b^{(2)}$
$b^{(2)}$
Serialization
$[h' W_{0,1}, \mathcal{W}_{0,2}, b_1^{(1)}, \hat{\mathcal{W}}^{(2)}, \mathcal{W}_{(2)}, b^{(2)}]$

Figure 4 Serialization example of quantized model parameters.

Only updates that improve at least one core metric (accuracy, precision, recall, or F1-score) without degrading any others are accepted. This logic is enforced deterministically by the *PoIm* contract, which governs model evolution and ensures that every change is auditable, trustless, and resistant to adversarial manipulation.

Every accepted or rejected update, along with its evaluation results, is permanently logged on-chain. This guarantees transparent, tamper-proof model provenance and allows peers to track, audit, or challenge any step of the training process. In contrast to centralized retraining, this decentralized micro-step protocol allows the model to evolve continuously and securely, driven entirely by peer contributions and verified in real time on-chain.

4.2 Inference Architecture

In our architecture, we support two cost-effective tiers for executing ML/DL inference over blockchain networks. Each tier provides a different trade-off between execution cost, verifiability, and decentralization.

On-chain Logic, Off-chain Execution (zero-cost). In this tier, the model parameters and execution logic are stored fully on-chain. However, the actual inference computation is performed off-chain by calling `view` functions, which are executed by any EVM client. This tier incurs zero gas cost for DeFi users or protocols while ensuring that the decision logic is derived directly from verifiable on-chain bytecode and model state. Since every node executes the same bytecode deterministically, outcomes are consistent and tamper-resistant. This design is ideal for platforms seeking lightweight classification with full code transparency and no execution fees.

Fully On-chain Inference (inference verifiable on-chain). Here, the inference is executed entirely on-chain as part of a state-modifying transaction, typically one that interacts with a DeFi protocol. The input is passed to the smart contract, which executes the classifier internally and enforces the classification result. This enables end-to-end verifiability and enforces decisions during transaction execution (e.g., rejecting or allowing access to protocol funds). While this incurs gas costs, our optimized design allows even moderately complex models, such as 10-layer CNNs with quantized integer arithmetic, to execute efficiently within Ethereum's gas limits. This tier is suited for scenarios like high TVL or security-critical DeFi functions, where classification must be verified on-chain without relying on off-chain entities to interpret results, as the outcome is enforced by on-chain consensus.

4.3 Layer-2 to Layer-1 Computation Separation

Model parameters are updated through decentralized training and governed by the Proof-of-Improvement (PoIm) protocol on Layer-2 (L2), where gas costs are significantly lower. These L2-validated model parameters must then be securely and accurately propagated to Layer 1 (L1) for use by inference contracts in either Tier 1 or Tier 2. To ensure that L1 inference contract parameters do not get tampered with, we employ a commit-verify propagation mechanism.

Commit to Model Hash (L2). Upon acceptance of a training update (yielding new parameters θ') by the *PoIm* protocol on L2, the contract computes a cryptographic hash `modelHash = keccak256(abi.encodePacked(`θ'`))`. This hash serves as a tamper-proof commitment to the new model.

Transmit Commitment (L2 → L1). The L2 contract sends this `modelHash` to L1 via the native L2-to-L1 bridge (e.g., Optimism's `L2ToL1MessagePasser`). The hash is recorded by the L1 contract that manages model updates.

Transmit Parameters and Verify (L2 → L1). In a subsequent transaction, the full model parameters θ' are transmitted. The L1 contract recomputes the hash and verifies it against the prior commitment. A mismatch results in the rejection of the update [59].

4.4 Formal Bit-Exact Verification

Weights are scaled by $S \in [10^6, 10^{18}]$ then packed into `int32[]` and up-cast to `int128` where safe. A fully-connected layer l executes $z^{(l)} = \mathrm{idiv}\big(\hat{W}^{(l)} z^{(l-1)} + \hat{b}^{(l)},\ S\big)$ If $\gamma > \Delta$, sign consistency holds for all validation inputs. For every compiled model, we prove $\forall x \in \mathbb{Z}^d$: $\mathcal{F}_{\text{on}}(x) = \mathcal{F}_{\text{off}}(x)$ under fixed-point scale S. We encode both paths as bit-vector (e.g., 256-bit) formulas and ask Z3 for `expr_on` $\neq$ `expr_off`. All models (linear, CNN, RNN) return `unsat`,

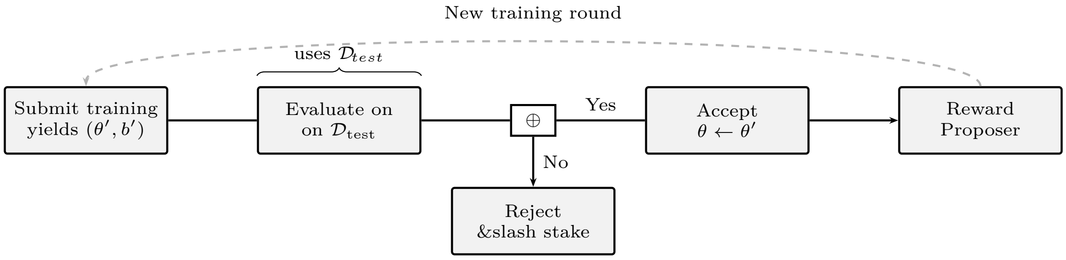
New training round
uses $\mathcal{D}_{test}$
Submit training yields (θ', b')
Evaluate on on $\mathcal{D}_{\text{test}}$
$\oplus$
Yes
No
Accept $\theta \leftarrow \theta'$
Reward Proposer
Reject &slash stake

Figure 5 *PoIm* protocol update flow. Each proposed model is evaluated on $\mathcal{D}_{\text{test}}$ and accepted only if it improves performance.

giving a machine-checked guarantee of equality. This design-time formal proof offers a strong guarantee that the compiled on-chain model faithfully implements the intended off-chain model logic under the specified fixed-point representation for all possible inputs. It ensures the intrinsic correctness of the model's translation to Solidity. This is distinct from, yet complementary to, the operational consistency checks performed during the lifecycle of the model on-chain. Listing 1 provides an example of a forward pass implemented in Solidity using fixed-point arithmetic.

Listing 1 Forward pass example for layer l using fixed-point arithmetic

```
for (uint i = 0; i < d_l; i++) {
    z[i] = bias_l[i];
    for (uint j = 0; j < d_{l-1}; j++) {
        z[i] += idiv(weights_l[i * d_{l-1} + j] * input[j], SCALE);
    }
}
```

Beyond the formal verification of the model logic itself, our protocol incorporates runtime consistency checks at critical junctures, such as model updates on L2, propagation to L1, and sample inferences, to ensure operational integrity in runtime.

Table 3 Analytic MAC bounds for CNN and RNN models with total input dimension $d = 3$.

Model	F	K	Bound
$\text{CNN}_{2\times2}$	2	2	1 714
$\text{CNN}_{4\times2}$	4	2	3 428
$\text{CNN}_{8\times3}$	8	3	4 832

(1)

Model	U	T	Bound
$\text{RNN}_{4\times2}$	4	2	7 064
$\text{RNN}_{8\times4}$	8	4	40 160

(2)

4.5 Gas Cost and Runtime Bound Analysis

Opcode Budget. For every multiply-accumulate (MAC) $a \leftarrow a + \frac{w \cdot x}{S}$ inside `classify`, the EVM executes (1) `SLOAD`$(w) = G_S = 100$ gas (warm), (2) `CALLDATALOAD`$(x) = G_C = 3$ gas, (3) `MUL` $= G_M = 5$ gas, (4) `DIV` $= G_D = 5$ gas, (5) `ADD` $= G_A = 3$ gas, (6) loop/stack bookkeeping $\approx G_L = 8$ gas.[1]

[1] Berlin fee schedule [21].

Hence, the cost per MAC is $g_{\text{MAC}} = G_S + G_C + G_M + G_D + G_A + G_L = 124$ gas. ReLU adds $G_R = 5$ gas per activation, and bias initialization costs $G_S + G_A = 103$ gas.

Linear Classifiers (LR / SVM). A linear model with d inputs executes one MAC per feature and adds a bias. Hence, $\boxed{G_{\text{LIN}}(d) = 124\,d + 103}$

For logistic regression, we approximate $\sigma(z)$ by a threshold on the logit, therefore no extra exponentiation is incurred; the cost matches SVM. With d=3 (example feature set) the bound gives $G_{\text{LIN}}(3) = 124 \times 3 + 103 = 475$ gas, which is three orders of magnitude below the deep models and negligible at call-sites.

CNN Bound. For an input of length d, kernel size K, and F filters, the convolution yields $o = d - K + 1$ positions and evaluates $F\,o\,K$ MACs; the fully connected read-out contributes a further $F\,o$ MACs. Thus, $\boxed{G_{\text{CNN}}(d, K, F) = g_{\text{MAC}}\,F\,o\,(K + 1) + G_R\,F\,o + 103\,F}$ (1).

RNN Bound. Let U be the hidden-state size, T the number of time steps, and $d_{\text{in}} = d/T$ the per-step input width. A gated update performs $U(d_{\text{in}} + U)$ MAC operations per time step. The total cost is $\boxed{G_{\text{RNN}}(d, U, T) = g_{\text{MAC}}\,T\,U(d_{\text{in}} + U + 1) + 15\,T\,U}$ (2). We fix the total input dimensionality $d = 3$ for both architectures. Table 3 shows the resulting gas bounds for several example CNN and RNN configurations.

4.6 Decentralized Model Update

Only models that show improvement in multiple evaluation metrics are accepted. No external oracle trust is required and all updates are verifiable on-chain within strict gas constraints.

Table 4 Notations used.

Symbol	Description
$\theta,\ \theta'$	current and proposed weight vectors
$b,\ b'$	current and proposed bias scalars
S	scaling constant, $S = 10^x$
$\mathcal{D}_{\text{test}} = \{(x_i, y_i)\}_{i=1}^n$	on-chain evaluation dataset
$x_i \in \mathbb{R}^d,\ y_i \in \{0, 1\}$	feature vector and label
$f_\theta(\cdot)$	classifier parameterized by (θ, b)
s	stake (in wei) deposited by proposer
$M, M', M_{\text{old}}, \widetilde{M}$	metric vectors before/after update
Δt	challenge window (e.g., 1-day)

Proof of Improvement (PoIm). Let f_θ denote the deployed classifier with an immutable architecture and on-chain weights θ (see Table 4 for notations). The core of our *PoIm* mechanism relies on $\mathcal{D}_{\text{test}} = \{(x_i, y_i)\}_{i=1}^n$, a canonical evaluation dataset stored directly on-chain. This dataset is intentionally curated to be compact, yet it is representative of critical attack vectors and desired model behaviors rather than being an exhaustive list of all historical transactions. Its manageable size is crucial for enabling efficient on-chain evaluation of proposed model updates within gas limits. Furthermore, to maintain its relevance and resist ossification, $\mathcal{D}_{\text{test}}$ itself is governed by a decentralized peer-based mechanism (e.g., DAO voting), allowing for agreed-upon additions, modifications, or removals of test samples over time (see Figure 5). Let θ' represent a submitted update. We define the classifier (i.e., linear) as $f_\theta(x) = \text{sign}\Big(\frac{1}{S}\sum_{i=1}^{d} \theta_i\, x_i \;+\; b\Big) \quad \in \{0, 1\}$. We define the evaluation function as $\text{Eval}(f_\theta, \mathcal{D}_{\text{test}}) \to (\text{Acc}, \text{F1}, \text{Prec}, \text{Rec})$, where all metrics are computed deterministically using Solidity logic and can be challenged.

New Training Submission. Users can propose a new model update by submitting a new training sample directly to the *PoIm* contract, which yields a change (if accepted) in new model weights θ' and biases b', and staking a minimum amount of tokens (or, e.g., any ERC20 token). Each submission must submit a stake, e.g., $s > 0$. The staked ERC20 serves as collateral for the proposal. If accepted, the contributor receives: $R = s + \sum_{k\in\{\text{Acc},\text{F1},\text{Prec},\text{Rec}\}} \alpha_k\,(M'_k - M_k)$. Each coefficient $\alpha_i \geq 0$ reflects the vault's value weighting for each metric improvement. For instance, if the vault has accumulated 1 ETH from failed update attempts, the payout R is proportionally distributed based on the magnitude of improvement across the four metrics.

Model Update Acceptance. An update θ' is accepted if it improves at least one core metric (accuracy, precision, recall, or F1-score) without degrading any of the other core metrics compared to the current model, based on on-chain evaluation over $\mathcal{D}_{\text{test}}$. The contract enforces this by ensuring that any accepted model update demonstrates improvement in at least one core metric without degrading others, thereby preventing overfitting a single target (e.g., maximizing recall while degrading precision). This improvement is verified on-chain on $\mathcal{D}_{\text{test}}$, which ensures that the proposed model performs better than the current one.

Test Set and Adversarial Update Rollback. We integrate an on-chain DAO mechanism that allows stakers (e.g, DeFi platforms) to collectively manage both model rollbacks and $\mathcal{D}_{\text{test}}$ test set updates. In our design, participants stake tokens to gain voting rights to propose and vote on critical actions such as adding or modifying test samples and reverting model weights (to previously committed L1 update) if suspicious behavior is observed (e.g., malicious training sample). In the case of a challenged update (i.e., one that introduces a loophole but still satisfies the acceptance criteria), a revealed update can be rolled back within a fixed period (e.g., <7 days). If the new model weights worsen in any performance metric compared to the previous metrics, the proposer loses their collateral and the proposal is discarded.

5 Evaluation

This section details our experimental setup, dataset construction, model configurations, and metrics used to evaluate our proposed framework for on-chain DeFi exploit detection and mitigation across various machine learning architectures. We also present a quantitative comparison against a baseline for on-chain ML.

5.1 Experimental Setup

All experiments were run on a machine equipped with an Intel Core i7 CPU and 16GB of RAM, running Windows 10. Key Python libraries we utilized include `scikit-learn` [39], `pandas` [45] (e.g., for data manipulation), `NumPy` [34] (e.g., for numerical operations), `web3.py` (e.g., for blockchain interaction), `py-solc-x` (for Solidity compilation), and the `Z3-solver` [17] (e.g., for formal verification logic).

For blockchain interactions, local L1 and L2 test environments were used. The L2 environment was configured using a Hardhat Network instance forking a live Optimism L2 rollup state (e.g., Optimism Mainnet), facilitating realistic gas calculations and execution

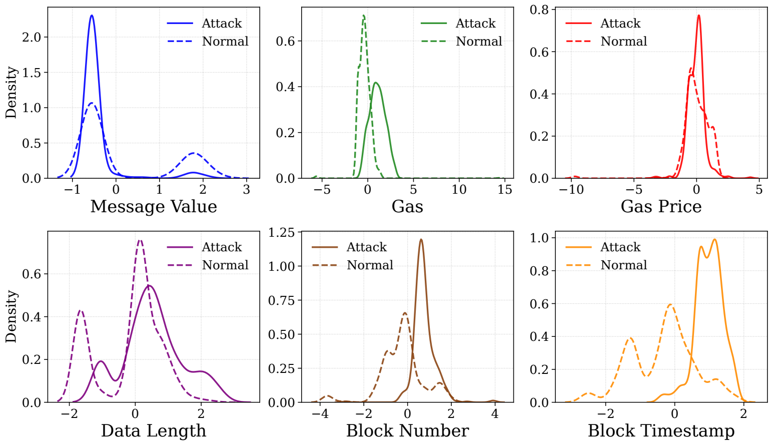

Figure 6 Distributions of normal and attack transactions.

behavior mirroring Optimism's characteristics. This L2 environment is primarily used for the decentralized training and *PoIm* mechanism. The L1 environment was simulated using Anvil[27], configured for persistence across experimental runs to reflect a stable mainnet-like chain.

Attack Dataset Construction. A significant challenge in evaluating DeFi exploit detection systems is the general absence of comprehensive, publicly available, raw transaction datasets suitable for behavioral modeling. While some prior work has focused on smart contract code analysis [69, 64, 66], readily usable transactional datasets for exploit detection are scarce. To address this, we undertook a meticulous manual collection and multi-stage verification process to construct a robust dataset for this study, covering attacks from 2020 to 2025. Our initial identification of potential exploits involved a broad survey of diverse sources, including industry security news (e.g., Rekt News [18]), detailed analyses on technical blogs, discussions on social media platforms (e.g., X), and curated public incident databases such as the DeFi Rekt Database [18] and DeFiLlama Exploits Dashboard [19]. Incidents and leads gathered from these channels, often further indexed or summarized by resources like DeFiHackLabs [58], guided our targeted retrieval of the specific transactions that executed each confirmed attack.

These transactions are primarily identified by their hashes on public blockchain explorers (e.g., Etherscan [22], Polygonscan [53]). All transactional records were independently retrieved and subsequently verified through direct blockchain queries via Web3 RPCs [14] to ensure authenticity and completeness of call data, receipts, and traces. The primary features extracted for model input include: `gas` (gas limit provided by sender), `block.timestamp`, `func_selector_encoded` (a label-encoded representation of the first four bytes of `msg.data`), `chain` ID (label-encoded), `msg.sender` (label-encoded), `tx.origin` (label-encoded), and `msg.to` (label-encoded), as shown in Figure 6. Numerical features were standardized, and categorical features were label-encoded. Features such as `block.timestamp` may overfit if the same timestamp is found in both training and testing. To mitigate this risk, we implemented a strict temporal train-test split. Training is exclusively on historical data, and testing is on future (unseen attacks). Therefore, the model generalizes better based on transaction behaviors rather than memorizing temporal artifacts. Transaction traces and contract interactions were analyzed to produce contextual annotations, including attack_name, links to exploited contract source files, incident report URLs, dates, identified root causes, and financial loss in USD (normalized to the time of exploit). This process was applied to transactions across multiple EVM-compatible blockchains, including *Ethereum, Binance Smart Chain (BSC), Polygon, Avalanche, Arbitrum, Fantom, Moonriver, and Base.* Table 5 presents an illustrative subset of these attributes.

Table 5 Example of a subset of fields from our exploit transaction dataset. Full transaction records include additional attributes such as detailed EVM context, blockchain metadata, and semantic annotations.

Field	**Example**	**Description**
tx_hash	0x78d7...2df4	Unique identifier of the exploit transaction
msg.sender	0x5aa5...30b7	Address that initiated the exploit
msg.value	0	ETH directly transferred in the transaction
gas_used	298210	Total gas consumed during execution
block.timestamp	1688462834	Unix timestamp of the exploit
root_cause	Unchecked external call	Vulnerability exploited in the contract
loss_USD	$1.94M	Financial loss via unauthorized token transfers
chain	Ethereum	Blockchain on which the exploit occurred

* *Only a subset of fields is shown here.*

We observed that an attack can manifest in a single atomic transaction (approximately 71% of the attacks in our dataset) or across multiple transactions. For multi-transaction attacks, all constituent transactions were grouped and assigned entirely to either the training or testing set to prevent data leakage and maintain the integrity of the attack sequence. The combined dataset of normal and attack transactions was then sorted chronologically by block.timestamp. Table 6 summarizes the distribution of the unique exploits that informed the construction of our dataset.

Table 6 Distribution of *unique* exploits (2020- 2025) and for DeFi financial losses.

Category	**Count**	**Loss (USD)**
Total distinct exploits	298	–
Exploits in training set	202	$1,877,229,549.86
Exploits in testing set	96	$1,858,400,900.66
Total exploits loss	–	**$3,735,630,450.52**

The resulting transaction data captures fine-grained, executable-level behaviors and intricate attacker-victim dynamics. This detailed resolution enables precise detection and classification of exploits and provides a foundation for exploring potential attack discovery. The final curated dataset of 298 unique attack vectors, corresponding to confirmed financial losses exceeding $3.74 billion (as per Table 6), combined with temporally aligned normal transactions, offers a rich and unique foundation for evaluating real-time defense mechanisms and rigorously training on-chain classifiers.

Model Architectures. Our framework was evaluated with diverse model architectures, including Multi-Layer Perceptrons (MLPs), Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and standard classifiers like Logistic Regression (LogReg), Support Vector Machines (SVM), and Decision Trees (DTs). All models utilized the 7 processed input features detailed previously. These features are observable by smart contracts during execution, enabling real-time attack detection. All models were evaluated with fixed-point parameters (scale 10^1 to 10^{18}).

Proof-of-Improvement (PoIm) Protocol. Each instance starts from baseline parameters and their performance metrics (Accuracy, F1-score, Precision, Recall) on a fixed test set ($\mathcal{D}_{\text{test}}$). An update is accepted only if it improves at least one metric without degrading others. Approved parameters are then eligible for propagation to L1 for inference.

Metrics. The performance of our framework was evaluated across three key dimensions: on-chain inference efficiency (gas costs, bytecode size, throughput, and prediction consistency), attack detection performance (accuracy, precision, recall, F1-score, and false positive rate), and model update mechanism costs (*PoIm* L2 update gas and L2-L1 propagation cost).

5.2 Performance of On-Chain DeFi Exploit Detection

We evaluate the effectiveness of on-chain machine learning classifiers in detecting previously unseen DeFi exploits. Each model was tested on a hold-out test set $\mathcal{D}_{\text{test}}$ comprising real-world attacks and benign transactions, temporally separated from the training set to simulate generalization to zero-day attacks. The unseen exploits in the test set include unique, protocol-specific attacks from various root causes that are absent from the training set.

We tested a diverse set of models: Logistic Regression (LogReg), Support Vector Machine (SVM), Decision Tree (DT), Multi-Layer Perceptron (MLP), 15 CNN variants (filters $F \in \{2, 4, 8, 10, 16\}$, kernel sizes $K \in \{1, 4, 5\}$), and RNNs with 8 units and with 1 and 7 timesteps (T=1 and T=7). All models utilized a shared 7-feature input vector, preprocessed via standardization and label-encoding as described in Section 5.1.

The fully trained CNN variants demonstrated strong detection capabilities, particularly in terms of recall. Many configurations achieved recall ≥ 0.96, with F1-scores generally ranging from $\sim$0.78 to 0.82, and precision reaching up to 0.8077 (CNN (F4, K1)). For instance, CNN(F4, K4) achieved a high accuracy of 0.9004 and an F1-score of 0.8200, preventing an estimated $1,857.6M in losses. This performance marks a significant improvement over any preliminary simulations where simpler CNN setups might have exhibited degenerate behavior.

RNN models also showed robust and balanced performance. Specifically, RNN(U=8, T=1) achieved an accuracy of 0.8517 with a high recall of 0.9792, contributing to $1,858.2M in prevented losses. The RNN(U=8, T=7) configuration maintained competitive performance, notably achieving higher precision (0.6607) and a lower false positive count (58 FPs) compared to RNN(U=8, T=1), with a slight trade-off in recall (0.9479).

Among other classifiers, LogReg and SVM performed well, with SVM achieving the highest AUC in this set (0.9739) and LogReg attaining perfect recall (1.0000). The Multi-Layer Perceptron (MLP) also demonstrated strong results, with an accuracy of 0.8665, F1-score of 0.7823, and a high recall of 0.9688. The DecisionTree, while achieving perfect recall, did so at the cost of a higher number of false positives (224 FPs), indicating overfitting to the attack class.

Table 7 On-Chain costs, sizes, and *PoIm* dynamics for LogReg, SVM, DT, MLP, CNN, and RNN models. **Ext. cl.:** inference by an EVM node.

Model	**Deploy L2** **(Gas)**	**Deploy L1** **(Gas)**	**L1 Inf. Gas** **(Gas)**	**L2 Size** **(Bytes)**	**L1 Size** **(Bytes)**	**L2-L1 (PoIm)** **(Gas)**	**Ext. cl.*** **($)**
LogReg	1,185,010	722,020	57,603	4,597	3,044	231,168	0
SVM	1,184,998	722,020	57,603	4,597	3,044	231,168	0
DT	878,284	435,490	33,414	3,352	1,707	96,489	0
MLP	2,137,045	1,116,415	138,173	7,266	4,866	5,754,732	0
CNN(F2, K1)	1,721,688	1,536,046	143,647	6,692	5,834	2,872,940	0
CNN(F4, K1)	2,134,766	1,949,124	244,284	6,692	5,834	4,885,680	0
CNN(F8, K1)	2,961,086	2,775,539	445,562	6,692	5,834	8,911,240	0
CNN(F10, K1)	3,374,357	3,188,810	546,202	6,692	5,834	10,924,040	0
CNN(F16, K1)	4,614,143	4,428,596	848,126	6,692	5,834	16,962,520	0
CNN(F2, K4)	1,721,688	1,536,046	158,856	6,692	5,834	3,177,120	0
CNN(F4, K4)	2,134,778	1,949,136	274,703	6,692	5,834	5,494,060	0
CNN(F8, K4)	2,961,098	2,775,551	506,397	6,692	5,834	10,127,940	0
CNN(F10, K4)	3,374,369	3,188,822	622,244	6,692	5,834	12,444,880	0
CNN(F16, K4)	4,614,167	4,428,620	969,788	6,692	5,834	19,395,760	0
CNN(F2, K5)	1,721,688	1,536,046	150,304	6,692	5,834	3,006,080	0
CNN(F4, K5)	2,134,778	1,949,136	257,598	6,692	5,834	5,151,960	0
CNN(F8, K5)	2,961,086	2,775,539	472,188	6,692	5,834	9,443,760	0
CNN(F10, K5)	3,374,345	3,188,798	579,482	6,692	5,834	11,589,640	0
CNN(F16, K5)	4,614,143	4,428,596	901,368	6,692	5,834	18,027,360	0
RNN(U8, T1)	3,520,668	3,338,663	561,791	7,413	6,571	11,235,820	0
RNN(U8, T7)	2,437,361	2,255,356	1,131,350	7,413	6,571	22,627,000	0

**Read-only RPC calls consume 0 gas on-chain.*

5.3 Efficiency, Cost, and Consistency

We analyze the on-chain operational costs, resource utilization, and behavioral consistency of the evaluated machine learning models. All quantitative data discussed refer to Table 7, which details L2 and L1 deployment gas, L1 inference gas, contract bytecode sizes, and key Proof-of-Improvement (PoIm) interaction costs where applicable.

Deployment and Inference Gas Costs. The on-chain footprint of simpler models like Logistic Regression (LogReg), Support Vector Machines (SVM), Decision Trees (DT), and a Multi-Layer Perceptron (MLP) showcases varying efficiencies. L2 *PoIm* contract deployment gas ranged from approximately 0.88M for the DT to 2.14M for the MLP, while their corresponding L1 inference contracts were lighter. Notably, L1 inference gas for LogReg, SVM, and DT was very low (33k–58k gas). The evaluated MLP required approximately 138k gas for L1 inference, still well within practical limits for on-chain execution.

Convolutional Neural Network (CNN) variants demonstrated a clear trend where both L2/L1 deployment gas and L1 inference gas scaled with architectural complexity, primarily driven by the number of filters (F) and kernel size (K). For instance, L2 deployment gas ranged from approximately 1.72M for CNN(F2, K1) up to 4.61M for larger configurations like CNN(F16, K1). Similarly, L1 inference gas for these CNNs varied from around 144k (CNN(F2, K1)) to over 969k (CNN(F16, K4)). Bytecode sizes for CNN L1 inference contracts were observed to be consistent for models sharing the same kernel size, as changes in filter count primarily affect the parameter size of the subsequent fully connected layer rather than the convolutional logic structure itself.

For the Recurrent Neural Network (RNN) models evaluated (both with U=8 units), the T=1 configuration (processing all 7 features in a single timestep) incurred higher L2 deployment gas (3.52M) compared to the T=7 configuration (processing 1 feature per 7 timesteps, 2.44M L2 gas). This difference is likely attributable to the larger input-to-hidden

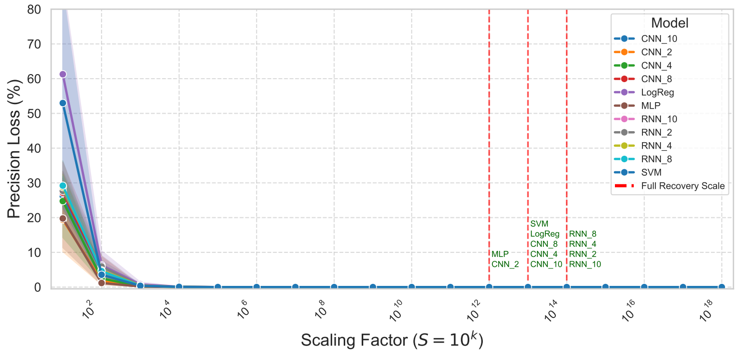

Figure 7 Bitwise consistency for different scaling factors of various models.

weight matrix (W_{xh}) in the RNN(U=8, T=1) model. Conversely, L1 inference gas was substantially higher for RNN(U=8, T=7) (1.13M gas) than for RNN(U=8, T=1) (0.56M gas), reflecting the increased number of recurrent steps executed on-chain for the sequence-based input.

The gas cost for transferring updated model parameters from an L2 *PoIm* contract to its L1 inference counterpart varied across model types. For SVM and DT, this L2-L1 update was relatively efficient (96k–231k gas). However, the LogReg and the dynamically trained MLP exhibited higher transfer costs (5.2M–5.7M gas, respectively), likely due to the encoding or size of their complete parameter sets being transferred. For the more complex CNN and RNN models, this L2-L1 parameter transfer for their (typically output layer) updates also showed significant gas consumption, ranging from 2.8M to over 22M gas, underscoring the cost implications of updating larger or more intricate models across layers.

Bitwise Consistency Verification. We evaluate empirically how fixed-point quantization affects on-chain inference consistency under varying scaling factors from 10^1 to 10^{18}, using MLP, logistic regression (LogReg), SVM, CNN, and RNN models (Figure 7). Precision loss diminishes rapidly as the scale increases, and full recovery (zero bitwise loss across all weights) is achieved at or above 10^{12} for all models. This confirms that fixed-point quantization, when aligned with sufficiently large scaling factors, can reliably preserve inference semantics on-chain across diverse architectures, which was also evaluated in terms of the models' performance.

***PoIm* Resilience Under Adversarial Training.** We present *PoIm* resistance against malicious updates via stress testing. We consider bootstrapping the decentralized model with 50 real training samples from both classes, and injecting 50% of the training data size as malicious or fabricated updates. These malicious updates either flip the class label or inject feature-noise. The 50 real samples are randomly selected from the original attack and normal transaction data to quantify how *PoIm* reacts when training samples degrade one or more metrics, and whether such poisoning affects performance. At each new training step, the update is re-evaluated on the full test set. If accepted by *PoIm*, the model is updated; otherwise, the current model is retained. We note that the bootstrapping phase influences how *PoIm* responds to subsequent updates. Overall, *PoIm* maintains more stable performance than the original (unfiltered) training process across all metrics on the same test data, as shown in Figure 8. Green vertical bands indicate accepted training samples, while reddish bands denote rejected ones. Notably, original linear models (e.g., logistic regression, SVM) suffer severe degradation without *PoIm* filtering.

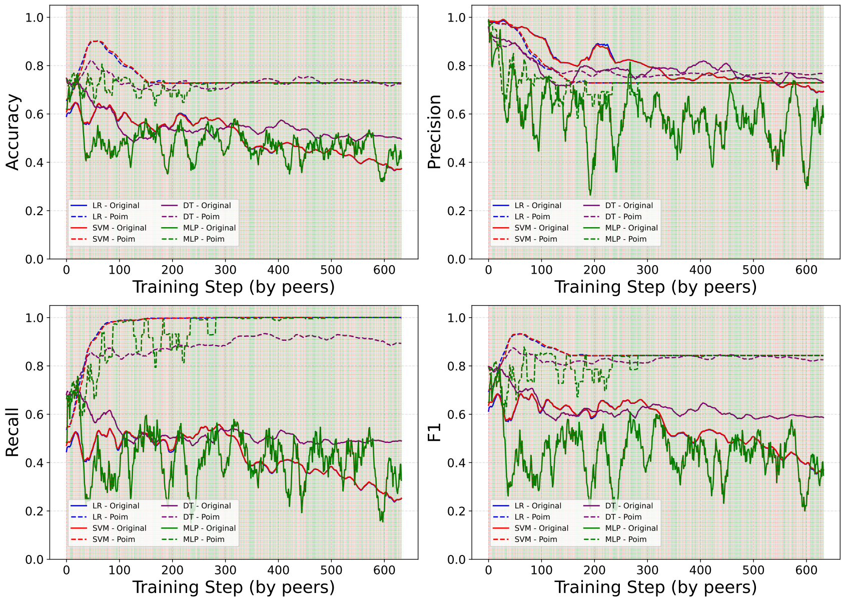

Figure 8 *PoIm* performance under 50% adversarial training samples. Green vertical lines indicate accepted updates. Red lines indicate rejected ones. *PoIm* stabilizes performance compared to the original training.

Inference Throughput. To evaluate real-time classification capability for the zero-cost `eth_call` tier, we measured inference throughput using our 10-layer CNN. Table 8 details these findings, which were obtained by sequentially querying a local Ethereum node. A single transaction is classified in approximately 68ms. While the average per-sample processing time for batches of 5 to 50 transactions stabilizes around 88-91ms, this performance enables a throughput of approximately 11 classifications per second by an off-chain monitoring entity. Such capacity is comparable to Ethereum's current average TPS (≈ 15), suitable for timely analysis.

Table 8 Inference throughput per second for (e.g., 10-layer) CNN for different transactions as batches. Ethereum's current TPS $\approx$ 15.

Batch Size	Total Time (s)	Avg Time/Sample (s)
1	0.0680	0.0680
5	0.3890	0.0778
10	0.9150	0.0915
20	1.7650	0.0883
50	4.5580	0.0912

5.4 Baseline Comparison

We compare our framework against ML2SC [42], which compiles MLP models into smart contracts for on-chain inference. While training in ML2SC is centralized, on-chain ML studies remain sparse. Our evaluation focuses on gas costs and contract behavior. We re-deployed all ML2SC MLP contracts, but observed that their design targets batch processing over a fixed internal dataset of 50 samples. Specifically, their `classify()` function returns an aggregate result (e.g., count of correct classifications), whereas our framework supports single-instance inference.

Table 9 Our framework vs. Ml2SC as a baselines [42].

Metric	**Our Framework**	**Baseline Method (ML2SC [42])**
Deployment Gas (Contract + Params)	1.07M - 2.17M gas	1.96M - 2.36M gas (contract code only)
Separate Parameter Setting Gas	0 gas	0.30M - 1.04M gas
Data Setting Gas	N/A	~**10.63M gas** (for 50 internal samples)
Total Setup Gas	**1.07M - 2.17M gas**	~**12.9M** - ~**14.0M gas**
Bytecode Size (Bytes)	~4,224 bytes	~9,425 - ~11,151 bytes
Inference Gas (classification on chain L1)	**81k - 261k gas (per call)**	~**2.0M** - ~**2.4M gas (for 50 internal samples)**
Output Type	Prediction (0 or 1)	Batch Result (e.g., count like 0 or 14)

Baseline `classify()` output (e.g., 14 (for MLP 1 layer and 1 neuron), 0 for others) indicates the number of correct predictions from its internal batch.

Deployment and setup costs differ significantly between our approach and the baseline. Our models embed parameters directly via the constructor and become operational with 1.07M–2.17M gas. In contrast, ML2SC baselines incur higher cumulative setup costs (~12.9M–14.0M gas) due to separate contract deployment (1.96M–2.36M gas), parameter setting (0.30M–1.04M gas), and dataset population (~10.63M gas for 50 samples), as detailed in Table 9. For inference, our framework supports efficient single-input predictions (81k–261k gas) with measurable throughput (~7.6–8.0 calls/sec), whereas baseline contracts consume ~2.0M–2.4M gas per call, reflecting batch evaluation of 50 internal samples. Their `view/pure classify()` functions are not suited for single-instance transactional inference. Finally, our contracts are smaller in size (~4,224 bytes) compared to baseline contracts (~9,425–11,151 bytes). The results demonstrate the efficiency and suitability of our framework for single-instance, real-time ML inference on-chain, especially in terms of gas cost and inference flexibility.

6 Discussion

Transaction Class Imbalance Mitigation. In our decentralized training, the model is updated incrementally by peers through submitted training samples. If normal transactions exceed attack samples by a large margin threshold (e.g., 5x), *PoIm* blocks further normal samples. Only attack samples are accepted until the balance is restored. This prevents skewed updates, which would decrease attack transaction detection accuracy.

Trust and Inference Verification. Our system is designed under the assumption that decentralized participants (i.e., DeFi platforms) perform training on an L2 network. However, we do not assume these participants are honest. To defend against adversarial training data injection, we incorporate per-sample verification and adversarial robustness checks at each training step. Specifically, after each submitted training sample, the *PoIm* evaluates the model against a fixed, immutable (agreed upon by peers) test set using four metrics: accuracy, precision, recall, and F1 score. The update is accepted only if it maintains or improves at least one of these metrics without degrading any others. Otherwise, the training step is discarded and the data point is excluded from the model. This method is effective in both cases: when malicious training samples are used or when honest user data fails to improve the model's performance. It ensures that even if malicious actors attempt to inject poisoned or manipulative data, their contributions cannot degrade the classifier's performance. Furthermore, all model training steps are conducted on L2 via a verifiable smart contract, and every accepted update is auditable through its associated state transition and event log. This guarantees transparency and accountability for each model state change. Once a model has reached a finalized state on L2 (e.g., by consensus or performance threshold), its quantized weights are serialized and committed via a cryptographic hash. This hash is propagated to L1 along with the raw weights. The L1 inference contract is a static model (non-trainable) that accepts the weights only if the hash matches, ensuring the integrity of the L2-originated model and protecting the L1 blockchain from tampering or weight substitution attacks. Thus, our architecture enforces correctness and robustness both during training (via metric-based rejection of adversarial updates) and during propagation (via hash-based integrity verification), with no reliance on external oracles, centralized validators.

Inference Tiers. Our two-tier inference architecture imposes a trade-off for protocol designers. The zero-cost tier uses `eth_call` execution. It can be used for protocols that do not need on-chain inference verifiability, yet get the on-chain verified data, such as wallet interfaces, warning users of potentially malicious transactions without incurring gas fees. For fully verified on-chain defense, protocols integrate the fully on-chain tier, which acts as a gatekeeper (e.g., IPS) by embedding the classification logic interface directly within a state-modifying transaction.

Future Work. The *PoIm* protocol only accepts updates that yield metric improvements. This approach might lead to convergence to a local optimum, where no single micro-update can further improve the model, even if a better global solution exists. Future work could explore other acceptance criteria, such as stochastic policies like simulated annealing [32], which would permit occasional slightly degrading steps to encourage broader exploration of the model space. Also, *PoIm*, similar to other stake-based governance systems, could be susceptible to centralization if voting rights, represented by tokens (e.g., linear voting), are openly tradable. A malicious actor could accumulate enough stake to influence the update of the test set or roll back stable updates. Mitigating this, for example, through quadratic voting or identity-based mechanisms such as know-your-customer (KYC) [23, 24], is a potential avenue for future work. Our framework assumes a decentralized and engaged set of participants (e.g., DeFi protocols themselves) whose long-term incentive is to ensure model integrity to protect individual DeFi protocols' funds. Finally, future work may study sophisticated adversarial strategies. This includes addressing game-theoretic risks such as front-running and detecting if latent backdoors in model updates exist.

7 Related Work

On-chain AI research seeks transparent, tamper-proof inference but faces the EVM's fixed-point arithmetic and gas limits [30, 44, 57, 42, 26]. Translators from ML to smart contract code such as ML2SC compile MLPs from PyTorch to Solidity, proving feasibility for small models yet incurring high gas per call on complex networks [57, 42]. For DeFi exploit detection, LookAhead [54], STING [67], and FlashGuard [5] inspect mempools or historical transactions to flag and mitigate attacks. Off-chain placement of these systems introduces latency [67], misses private-relay transactions, and introduces centralized control. Off-chain ML with on-chain verification is another direction of research. zkML [11], for example, attaches zk-SNARK [51] proofs to each inference, preserving privacy but multiplying compute and memory requirements by orders of magnitude [49, 65, 29].

Our cryptographic verification is distinct from zkML systems. ZKML frameworks use zero-knowledge proofs (e.g., zk-SNARKs) to verify that a specific computation, such as an ML inference, was executed honestly with a private model [11]. This provides computational integrity but has substantial overhead. For instance, ZKML needs powerful hardware (up to 1TB of RAM for a distilled GPT-2 model) and can have proving times of nearly an hour [11].

In contrast, our framework employs a much simpler and more gas-efficient commit-verify scheme. Our approach prioritizes provenance and data integrity over computational privacy since the DeFi attacks are public. This makes our approach practical for low-cost, real-time use and suitable for mitigating DeFi attacks.

opML [15] (fraud-proof) treats results as valid unless a verifier proves otherwise, reducing prover cost at the price of economic guarantees. However, it does not provide cryptographic security [49, 16]. Agatha applies similar fraud proofs to DNNs on Ethereum [70]. Proposals for decentralized model marketplaces, federated learning with ZK privacy, and DAO-based model governance [7] either offload heavy compute or evaluate on limited node sets [15]. In contrast, our approach is end-to-end, fully audited inference while remaining within L2 and L1 mainnet gas limits.

8 Conclusion

We presented a fully decentralized and verifiable, on-chain ML/DL framework for real-time DeFi exploit detection. Our approach enables classification of transactions at execution time using a deterministic, gas-free inference mechanism embedded in smart contracts. We proposed Proof-of-Improvement (PoIm), a decentralized, stake-based model update protocol that accepts only provably superior updates. The system guarantees inference consistency, bounded gas usage, and resistance to adversarial submissions. Empirical evaluation on 298 real-world DeFi exploits indicates high detection performance and practical feasibility. This work establishes a new model for integrating ML-driven defenses into DeFi protocols with minimal latency, overhead, and maximal decentralization.

References

1 Hedgey finance. https://hedgey.finance/, 2024. Accessed: 2024-05-08.

2 Martín Abadi. Tensorflow: learning functions at scale. In *Proceedings of the 21st ACM SIGPLAN international conference on functional programming*, pages 1–1, 2016.

3 Sa'ed Abed, Reem Jaffal, Bassam J Mohd, and Mohammad Al-Shayeji. An analysis and evaluation of lightweight hash functions for blockchain-based iot devices. *Cluster computing*, 24:3065–3084, 2021. doi:10.1007/S10586-021-03324-1.

4 Alchemy. Web3 Development Platform. https://www.alchemy.com, 2024.

5 Abdulrahman Alhaidari, Balaji Palanisamy, and Prashant Krishnamurthy. Poster: Flashguard: Real-time disruption of non-price flash loan attacks in defi. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24)*, page 3, Salt Lake City, UT, USA, october 14–18 2024. ACM. doi:10.1145/3658644.3691385.

6 Abdulrahman Alhaidari, Balaji Palanisamy, and Prashant Krishnamurthy. Protecting defi platforms against non-price flash loan attacks. In *Proceedings of the Fifteenth ACM Conference on Data and Application Security and Privacy (CODASPY '25)*, pages 1–12, Pittsburgh, PA, USA, 2025. ACM. doi:10.1145/3714393.3726503.

7 Dana Alsagheer, Lei Xu, and Weidong Shi. Decentralized machine learning governance: Overview, opportunities, and challenges. *IEEE Access*, 11:96718–96732, 2023. doi:10.1109/ACCESS.2023.3311713.

8 Raphael Auer, Bernhard Haslhofer, Stefan Kitzler, Pietro Saggese, and Friedhelm Victor. The technology of decentralized finance (defi). *Digital Finance*, 6(1):55–95, 2024.

9 Nic Carter and Linda Jeng. Defi protocol risks: The paradox of defi. *Regtech, suptech and beyond: innovation and technology in financial services" riskbooks–forthcoming Q*, 3, 2021.

10 Stefanos Chaliasos, Marcos Antonios Charalambous, Liyi Zhou, Rafaila Galanopoulou, Arthur Gervais, Dimitris Mitropoulos, and Benjamin Livshits. Smart contract and defi security tools: Do they meet the needs of practitioners? In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, pages 1–13, 2024. doi:10.1145/3597503.3623302.

11 Bing-Jyue Chen, Suppakit Waiwitlikhit, Ion Stoica, and Daniel Kang. Zkml: An optimizing system for ml inference in zero-knowledge proofs. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pages 560–574, 2024. doi:10.1145/3627703.3650088.

12 Huashan Chen, Marcus Pendleton, Laurent Njilla, and Shouhuai Xu. A survey on ethereum systems security: Vulnerabilities, attacks, and defenses. *ACM Computing Surveys (CSUR)*, 53(3):1–43, 2020. doi:10.1145/3391195.

13 Arka Rai Choudhuri, Sanjam Garg, Julien Piet, and Guru-Vamsi Policharla. Mempool privacy via batched threshold encryption: Attacks and defenses. *Cryptology ePrint Archive*, 2024.

14 Web3.js Contributors. Web3.js, 2024. URL: https://github.com/web3/web3.js.

15 KD Conway, Cathie So, Xiaohang Yu, and Kartin Wong. opml: Optimistic machine learning on blockchain. *arXiv preprint arXiv:2401.17555*, 2024. doi:10.48550/arXiv.2401.17555.

16 Sourav Das, Vinay Joseph Ribeiro, and Abhijeet Anand. Yoda: Enabling computationally intensive contracts on blockchains with byzantine and selfish nodes. *arXiv preprint arXiv:1811.03265*, 2018. arXiv:1811.03265.

17 Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

18 De.Fi. Rekt db, 2024. Accessed: 2024-06-07. URL: https://de.fi/rekt-database.

19 DeFiLlama. DeFi Dashboard. https://defillama.com, 2024. Accessed: 2024-05-14.

20 Kaustubh Dwivedi, Ankit Agrawal, Ashutosh Bhatia, and Kamlesh Tiwari. A novel classification of attacks on blockchain layers: Vulnerabilities, attacks, mitigations, and research directions. *arXiv preprint arXiv:2404.18090*, 2024. doi:10.48550/arXiv.2404.18090.

21 Ethereum Foundation. Ethereum improvement proposals. https://eips.ethereum.org/, 2025.

22 Etherscan. Ethereum blockchain explorer. http://etherscan.io/, 2024.

23 Andres Fabrega, Amy Zhao, Jay Yu, James Austgen, Sarah Allen, Kushal Babel, Mahimna Kelkar, and Ari Juels. Voting-Bloc Entropy: A New Metric for DAO Decentralization. In *2025 USENIX Security Symposium (USENIX Security 25)*, 2025.

24 Rainer Feichtinger, Robin Fritsch, Lioba Heimbach, Yann Vonlanthen, and Roger Wattenhofer. Sok: Attacks on daos. In *6th Conference on Advances in Financial Technologies (AFT 2024)*, pages 28–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.AFT.2024.28.

25 Flashbots. Mitigating MEV in Blockchain. https://www.flashbots.net, 2024. Accessed: 2024-07-23.

26 Ethereum Foundation. Ethereum, 2024. URL: https://ethereum.org/.

27 Foundry. Blazing fast, portable and modular toolkit for Ethereum application development written in Rust. https://getfoundry.sh, 2024.

28 Rundong Gan, Liyi Zhou, Le Wang, Kaihua Qin, and Xiaodong Lin. Defialigner: Leveraging symbolic analysis and large language models for inconsistency detection in decentralized finance. In *6th Conference on Advances in Financial Technologies (AFT 2024)*, pages 7–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.AFT.2024.7.

29 Bianca-Mihaela Ganescu and Jonathan Passerat-Palmbach. Trust the process: Zero-knowledge machine learning to enhance trust in generative ai interactions. *arXiv preprint*, 2024. doi:10.48550/arXiv.2402.06414.

30 Caleb Geren, Amanda Board, Gaby G Dagher, Tim Andersen, and Jun Zhuang. Blockchain for large language model security and safety: A holistic survey. *ACM SIGKDD Explorations Newsletter*, 26(2):1–20, 2025. doi:10.1145/3715073.3715075.

31 Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 3–16, 2016. doi:10.1145/2976749.2978341.

32 Thomas Guilmeau, Emilie Chouzenoux, and Víctor Elvira. Simulated annealing: A review and a new scheme. In *2021 IEEE statistical signal processing workshop (SSP)*, pages 101–105. IEEE, 2021. doi:10.1109/SSP49050.2021.9513782.

33 Yang Guo. A review of machine learning-based zero-day attack detection: Challenges and future directions. *Computer communications*, 198:175–185, 2023. doi:10.1016/J.COMCOM.2022.11.001.

34 Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020. doi:10.1038/S41586-020-2649-2.

35 Ningyu He, Lei Wu, Haoyu Wang, Yao Guo, and Xuxian Jiang. Characterizing code clones in the ethereum smart contract ecosystem. In *International Conference on Financial Cryptography and Data Security*, pages 654–675. Springer, 2020. doi:10.1007/978-3-030-51280-4_35.

36 Lioba Heimbach and Roger Wattenhofer. Eliminating sandwich attacks with the help of game theory. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pages 153–167, 2022. doi:10.1145/3488932.3517390.

37 Safak Kayikci and Taghi M Khoshgoftaar. Blockchain meets machine learning: a survey. *Journal of Big Data*, 11(1):9, 2024. doi:10.1186/S40537-023-00852-Y.

38 Nikhil Ketkar. Introduction to keras. In *Deep learning with python: a hands-on introduction*, pages 97–111. Springer, 2017.

39 Oliver Kramer and Oliver Kramer. Scikit-learn. *Machine learning for evolution strategies*, pages 45–53, 2016.

40 Wenkai Li, Xiaoqi Li, Yuqing Zhang, and Zongwei Li. Defitail: Defi protocol inspection through cross-contract execution analysis. In *Companion Proceedings of the ACM on Web Conference 2024*, pages 786–789, 2024. doi:10.1145/3589335.3651488.

41 Xiaofan Li, Jin Yang, Jiaqi Chen, Yuzhe Tang, and Xing Gao. Characterizing ethereum upgradable smart contracts and their security implications. In *Proceedings of the ACM Web Conference 2024*, pages 1847–1858, 2024. doi:10.1145/3589334.3645640.

42 Zhikai Li, Steve Vott, and Bhaskar Krishnamachari. Ml2sc: Deploying machine learning models as smart contracts on the blockchain. In *2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 645–649. IEEE, 2024. doi:10.1109/ICBC59979.2024.10634431.

43 Yulin Liu, Yuxuan Lu, Kartik Nayak, Fan Zhang, Luyao Zhang, and Yinhong Zhao. Empirical analysis of eip-1559: Transaction fees, waiting times, and consensus security. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2099–2113, 2022. doi:10.1145/3548606.3559341.

44 Rischan Mafrur. Ai-based crypto tokens: The illusion of decentralized ai? *IET Blockchain*, 5(1):e70015, 2025. doi:10.1049/BLC2.70015.

45 Wes McKinney et al. pandas: a foundational python library for data analysis and statistics. *Python for high performance and scientific computing*, 14(9):1–9, 2011.

46 Johnnatan Messias, Vabuk Pahari, Balakrishnan Chandrasekaran, Krishna P Gummadi, and Patrick Loiseau. Dissecting bitcoin and ethereum transactions: On the lack of transaction contention and prioritization transparency in blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 221–240. Springer, 2023. doi:10.1007/978-3-031-47751-5_13.

47 Nexus Mutual. CREAM Finance Hack. https://nexusmutual.io/claims-stories/cream-finance-hack, 2025. Accessed: 2025-02-05.

48 Optimism Foundation. Optimism. https://www.optimism.io/, 2025. Accessed: 2025-02-21.

49 Zhizhi Peng, Taotao Wang, Chonghe Zhao, Guofu Liao, Zibin Lin, Yifeng Liu, Bin Cao, Long Shi, Qing Yang, and Shengli Zhang. A survey of zero-knowledge proof based verifiable machine learning. *arXiv preprint arXiv:2502.18535*, 2025.

50 Daniel Perez and Benjamin Livshits. Smart contract vulnerabilities: Vulnerable does not imply exploited. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1325–1341, 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/perez.

51 Maksym Petkus. Why and how zk-snark works. *arXiv preprint arXiv:1906.07221*, 2019. arXiv:1906.07221.

52 Valentina Piantadosi, Giovanni Rosa, Davide Placella, Simone Scalabrino, and Rocco Oliveto. Detecting functional and security-related issues in smart contracts: A systematic literature review. *Software: Practice and experience*, 53(2):465–495, 2023. doi:10.1002/SPE.3156.

53 PolygonScan. Polygon Gas Tracker. https://polygonscan.com/gastracker, 2024. Accessed: 2024-08-30.

54 Shoupeng Ren, Lipeng He, Tianyu Tu, Di Wu, Jian Liu, Kui Ren, and Chun Chen. Lookahead: Preventing defi attacks via unveiling adversarial contracts. *arXiv preprint arXiv:2401.07261*, 2024.

55 Sarwar Sayeed, Hector Marco-Gisbert, and Tom Caira. Smart contract: Attacks and protections. *Ieee Access*, 8:24416–24427, 2020. doi:10.1109/ACCESS.2020.2970495.

56 Cosimo Sguanci, Roberto Spatafora, and Andrea Mario Vergani. Layer 2 blockchain scaling: A survey. *arXiv preprint arXiv:2107.10881*, 2021. arXiv:2107.10881.

57 Nikumbh Sarthak Sham, Sandip Chakraborty, and Shamik Sural. Generation of optimized solidity code for machine learning models using llms. *arXiv preprint arXiv:2503.06203*, 2025. doi:10.48550/arXiv.2503.06203.

58 SunWeb3Sec. DeFiHackLabs. https://github.com/SunWeb3Sec/DeFiHackLabs, 2024. Accessed: 2024-05-08.

59 Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains. In *Aspects of Computation and Automata Theory with Applications*, pages 377–424. World Scientific, 2024.

60 Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. Blockchain scaling using rollups: A comprehensive survey. *IEEE Access*, 10:93039–93054, 2022. doi:10.1109/ACCESS.2022.3200051.

61 Xiaojie Wang, Hanxue Li, Ling Yi, Zhaolong Ning, Song Guo, and Yan Zhang. A survey on off-chain networks: Frameworks, technologies, solutions and challenges. *arXiv preprint arXiv:2311.10298*, 2023. doi:10.48550/arXiv.2311.10298.

62 Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

63 Zihuan Xu and Lei Chen. L2chain: Towards high-performance, confidential and secure layer-2 blockchain solution for decentralized applications. *Proceedings of the VLDB Endowment*, 16(4):986–999, 2022. doi:10.14778/3574245.3574278.

64 Chavhan Sujeet Yashavant, Saurabh Kumar, and Amey Karkare. Scrawld: A dataset of real world ethereum smart contracts labelled with vulnerabilities. *arXiv preprint arXiv:2202.11409*, 2022. arXiv:2202.11409.

65 Zheming Ye, Xiaodong Qi, Zhao Zhang, and Cheqing Jin. Yoimiya: A scalable framework for optimal resource utilization in zk-snark systems. *arXiv preprint arXiv:2502.18288*, 2025. doi:10.48550/arXiv.2502.18288.

66 Pengcheng Zhang, Feng Xiao, and Xiapu Luo. A framework and dataset for bugs in ethereum smart contracts. In *2020 IEEE international conference on software maintenance and evolution (ICSME)*, pages 139–150. IEEE, 2020. doi:10.1109/ICSME46990.2020.00023.

67 Zhuo Zhang, Zhiqiang Lin, Marcelo Morales, Xiangyu Zhang, and Kaiyuan Zhang. Your exploit is mine: Instantly synthesizing counterattack smart contract. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1757–1774, 2023. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/zhang-zhuo-exploit.

68 Zhuo Zhang, Brian Zhang, Wen Xu, and Zhiqiang Lin. Demystifying exploitable bugs in smart contracts. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 615–627. IEEE, 2023. doi:10.1109/ICSE48619.2023.00061.

69 Zibin Zheng, Jianzhong Su, Jiachi Chen, David Lo, Zhijie Zhong, and Mingxi Ye. Dappscan: building large-scale datasets for smart contract weaknesses in dapp projects. *IEEE Transactions on Software Engineering*, 2024.

70 Zihan Zheng, Peichen Xie, Xian Zhang, Shuo Chen, Yang Chen, Xiaobing Guo, Guangzhong Sun, Guangyu Sun, and Lidong Zhou. Agatha: Smart contract for dnn computation. *arXiv preprint arXiv:2105.04919*, 2021. arXiv:2105.04919.

71 Liyi Zhou, Xihan Xiong, Jens Ernstberger, Stefanos Chaliasos, Zhipeng Wang, Ye Wang, Kaihua Qin, Roger Wattenhofer, Dawn Song, and Arthur Gervais. Sok: Decentralized finance (defi) attacks. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2444–2461. IEEE, 2023. doi:10.1109/SP46215.2023.10179435.