Smart-LLaMA-DPO: Reinforced Large Language Model for Explainable Smart Contract Vulnerability Detection

LEI YU, Institute of Software, Chinese Academy of Sciences, China
ZHIRONG HUANG, Institute of Software, Chinese Academy of Sciences, China
HANG YUAN, Institute of Software, Chinese Academy of Sciences, China
SHIQI CHENG, Institute of Software, Chinese Academy of Sciences, China
LI YANG*, Institute of Software, Chinese Academy of Sciences, China
FENGJUN ZHANG*, Institute of Software, Chinese Academy of Sciences, China
CHENJIE SHEN, Institute of Software, Chinese Academy of Sciences, China
JIAJIA MA, Institute of Software, Chinese Academy of Sciences, China
JINGYUAN ZHANG, Institute of Software, Chinese Academy of Sciences, China
JUNYI LU, Institute of Software, Chinese Academy of Sciences, China
CHUN ZUO, Sinosoft Company Limited, China

Smart contract vulnerability detection is a critical challenge in the rapidly evolving blockchain landscape. Existing vulnerability detection methods face two main issues: (1) Existing datasets lack comprehensiveness and sufficient quality, with limited vulnerability type coverage and insufficient distinction between high-quality and low-quality explanations for preference learning. (2) Large language models (LLMs) often struggle with accurately interpreting specific concepts in smart contract security. Through our empirical analysis, we found that even after continual pre-training and supervised fine-tuning, LLMs still exhibit limitations in precisely understanding the execution order of state changes in smart contracts, which can lead to incorrect vulnerability explanations despite making correct detection decisions. These limitations result in poor detection performance, leading to potentially severe financial losses. To address these challenges, we propose Smart-LLaMA-DPO, an advanced detection method based on the LLaMA-3.1-8B. First, we construct a comprehensive dataset covering

*Corresponding authors.
Lei Yu, Zhirong Huang, Hang Yuan, Chenjie Shen, Jingyuan Zhang, Junyi Lu are also affiliated with University of Chinese Academy of Sciences, China.
Lei Yu, Zhirong Huang, Fengjun Zhang, Jiajia Ma, Jingyuan Zhang are also affiliated with Integrative Innovation Center, Institute of Software, Chinese Academy of Sciences, China.
Lei Yu, Zhirong Huang, Fengjun Zhang, Jingyuan Zhang are also affiliated with Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences.
Hang Yuan, Shiqi Cheng, Li Yang, Chenjie Shen, Junyi Lu are also affiliated with Laboratory of Precise Computing, Institute of Software, Chinese Academy of Sciences.

Authors' Contact Information: Lei Yu, Institute of Software, Chinese Academy of Sciences, Beijing, China, yulei2022@iscas.ac.cn; Zhirong Huang, Institute of Software, Chinese Academy of Sciences, Beijing, China, huangzhirong2022@iscas.ac.cn; Hang Yuan, Institute of Software, Chinese Academy of Sciences, Beijing, China, yuanhang2023@iscas.ac.cn; Shiqi Cheng, Institute of Software, Chinese Academy of Sciences, Beijing, China, chengshiqi@iscas.ac.cn; Li Yang*, Institute of Software, Chinese Academy of Sciences, Beijing, China, yangli2017@iscas.ac.cn; Fengjun Zhang*, Institute of Software, Chinese Academy of Sciences, Beijing, China, fengjun@iscas.ac.cn; Chenjie Shen, Institute of Software, Chinese Academy of Sciences, Beijing, China, shenchenjie22@mails.ucas.ac.cn; Jiajia Ma, Institute of Software, Chinese Academy of Sciences, Beijing, China, majiajia@iscas.ac.cn; Jingyuan Zhang, Institute of Software, Chinese Academy of Sciences, Beijing, China, zhangjingyuan2023@iscas.ac.cn; Junyi Lu, Institute of Software, Chinese Academy of Sciences, Beijing, China, lujunyi2022@iscas.ac.cn; Chun Zuo, Sinosoft Company Limited, Beijing, China, zuochun@sinosoft.com.cn.

Please use nonacm option or ACM Engage class to enable CC licenses
This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2025 Copyright held by the owner/author(s).
ACM 2994-970X/2025/7-ARTISSTA009
https://doi.org/10.1145/3728878

four vulnerability types and machine-unauditable vulnerabilities, containing labels, detailed explanations, and precise vulnerability locations for Supervised Fine-Tuning (SFT), as well as paired high-quality and low-quality outputs for Direct Preference Optimization (DPO). Second, we perform continual pre-training using large-scale smart contract code to enhance the LLM's understanding of specific security practices in smart contracts. Futhermore, we conduct supervised fine-tuning with our comprehensive dataset. Finally, we apply DPO, which leverages human feedback to improve the quality of generated explanations. Smart-LLaMA-DPO utilizes a specially designed loss function that encourages the LLM to increase the probability of preferred outputs while decreasing the probability of non-preferred outputs, thereby enhancing the LLM's ability to generate high-quality explanations. We evaluate Smart-LLaMA-DPO on four major vulnerability types: reentrancy, timestamp dependence, integer overflow/underflow, and delegatecall, as well as machine-unauditable vulnerabilities. Our method significantly outperforms state-of-the-art baselines, with average improvements of 10.43% in F1 score and 7.87% in accuracy. Moreover, both LLM evaluation and human evaluation demonstrate the superior quality of explanations generated by Smart-LLaMA-DPO in terms of correctness, thoroughness, and clarity.

Additional Key Words and Phrases: Smart Contract, Large Language Models, Direct Preference Optimization

ACM Reference Format:
Lei Yu, Zhirong Huang, Hang Yuan, Shiqi Cheng, Li Yang*, Fengjun Zhang*, Chenjie Shen, Jiajia Ma, Jingyuan Zhang, Junyi Lu, and Chun Zuo. 2025. Smart-LLaMA-DPO: Reinforced Large Language Model for Explainable Smart Contract Vulnerability Detection. *Proc. ACM Softw. Eng.* 2, ISSTA, Article ISSTA009 (July 2025), 24 pages. https://doi.org/10.1145/3728878

1 Introduction

Blockchain technology has been rapidly adopted across various domains due to its decentralized architecture [55]. This innovative technology enables the creation of secure, distributed digital ledgers for recording transactions [24]. By utilizing advanced cryptographic methods, blockchain ensures the integrity and verification of each transaction[67, 72]. Within this ecosystem, smart contracts function as self-executing programs on the blockchain, automating the management of digital assets such as cryptocurrencies. These contracts are activated when specific conditions are met and, once deployed, become permanent components of the blockchain [77]. However, the immutability and inherent complexity of smart contracts pose significant security challenges [77]. The well-known DAO incident [14, 35] serves as a cautionary example, demonstrating the potential severity of such vulnerabilities. This security breach resulted in the illegal transfer of $60 million worth of Ethereum, causing widespread impact on the blockchain community [2, 23].

Researchers have developed various techniques for smart contract vulnerability detection, such as symbolic execution (Oyente [33], Mythril [37]) and static analysis (Slither [19], SmartCheck [56]). However, these methods often rely on predefined patterns and perform poorly in complex scenarios. Qian et al. [46] proposed cross-modality learning approaches that leverage information from both bytecode and source code. Recently, researchers have begun to explore the potential of LLMs in smart contract vulnerability detection and explanation (iAudit [34]). While iAudit shows promise in detecting logic vulnerabilities, its two-stage architecture that separates detection and explanation tasks can lead to inconsistencies. For instance shown in Fig. 3, the detector incorrectly identifies a reentrancy vulnerability in the contract and misinterprets the sequence of external calls and state changes. While the Price Oracle Manipulation vulnerability is correctly identified, it is incorrectly attributed to insufficient access control rather than the critical issue of reliance on the Uniswap pool's instant price. Such inconsistencies, although mitigated by iAudit's Ranker-Critic mechanism, could still affect developers' remediation decisions. We conducted a systematic evaluation of existing smart contract vulnerability datasets, as shown in Table 1. While most existing datasets (A-D) provide only basic vulnerability labels without detailed explanations or location information, iAudit made progress by including these aspects. However, iAudit's dataset primarily

focuses on logic vulnerabilities, and its data enhancement process relies on LLMs without human expert verification after LLM generation. Furthermore, the current datasets lack the structured format to support Direct Preference Optimization (DPO) training [48], which significantly affects the LLM's ability to generate high-quality vulnerability explanations that match human expert standards.

Beyond iAudit, general LLMs also face significant challenges in the domain of smart contract vulnerability detection. They often encounter difficulties in dealing with smart contract-specific concepts and security implications. As shown in Fig. 1, when faced with two explanations, one being an incorrect explanation from a general LLM (LLaMA3.1-8B-Instruct) and the other a partially correct explanation from Smart-LLaMA-DPO (only after continual pre-training and supervised fine-tuning), the general LLM incorrectly identifies a reentrancy vulnerability in the contract. It misinterprets the meaning of external calls in the 'buyInternal()' function, failing to recognize that reentrancy vulnerabilities typically occur when contract state or balance changes are made after external calls, which is not the case in this contract. Smart-LLaMA-DPO (only after continual pre-training and supervised fine-tuning) correctly identifies the contract's security but still misunderstands the order of operations in its explanation. This subtle but critical misunderstanding highlights the limitations of relying solely on continual pre-training and supervised fine-tuning.

To address these challenges, we propose Smart-LLaMA-DPO, based on the LLaMA-3.1-8B model. This approach combines continual pre-training, supervised fine-tuning, and direct preference optimization (DPO). Unlike iAudit's two-stage architecture, which may lead to inconsistencies between detection and explanation, our approach leverages Direct Preference Optimization (DPO) [48] to align these tasks within a unified framework, enabling more consistent and context-aware results. By ultilizing large-scale smart contract code for domain-specific pre-training, we enable the LLM to better understand the syntax and semantics of smart contracts, overcoming the limitations of general LLMs in understanding contract semantics as shown in Fig. 1. Subsequently, we construct a comprehensive smart contract vulnerability dataset containing detailed explanations and precise location information. The fine-tuning process utilizing this high-quality dataset enhances the LLM's capabilities, enabling it to not only detect vulnerabilities but also generate explanations. This approach effectively addresses the limitations of existing datasets as illustrated in Table 1. Finally, to address the issues in explanation quality that persist even after CPT and SFT training (as seen in Smart-LLaMA-DPO (only after CPT and SFT)'s partially incorrect explanation in Fig. 1), we innovatively introduce DPO [48]. By constructing a dataset containing pairs of model outputs with varying quality, DPO enables the LLM to learn to generate higher quality explanations that better align with human expert expectations. Each pair in this dataset consists of two outputs: one preferred by human experts, representing high-quality explanations, and another of lower quality. The core of the Smart-LLaMA-DPO approach lies in learning directly from expert preferences, without the need for explicit reward modeling or complex reinforcement learning processes. Smart-LLaMA-DPO uses a specially designed loss function that encourages the LLM to increase the probability of preferred outputs while decreasing the probability of non-preferred outputs.

In the data construction process, we used Qwen2.5-72B-Instruct and Mistral-Large-Instruct-2407-123B to generate initial explanations, scored by Llama-3.1-70B-Instruct. Smart contract security experts refined high-scoring explanations to create the SFT dataset. For the DPO dataset, experts rewrote lower-scored outputs into "suboptimal" versions, which, while correct, lack depth and clarity compared to high-quality outputs, creating a reasonable quality gap.

To validate the effectiveness of Smart-LLaMA-DPO, we conducted a comprehensive experimental evaluation. We evaluated the Smart-LLaMA-DPO framework on a challenging dataset [46] covering four major vulnerability types (reentrancy, timestamp dependence, integer overflow/underflow, and delegatecall) and machine unauditable vulnerabilities (seven types) from [73]. The results show that

Smart-LLaMA-DPO significantly outperforms state-of-the-art methods across all vulnerability types. Smart-LLaMA-DPO achieves F1 scores 7.51%, 1.54%, 11.07%, and 6.06% higher than the previous best performance (DMT) for reentrancy, timestamp dependence, integer overflow/underflow, and delegatecall vulnerabilities, respectively. In terms of accuracy, Smart-LLaMA-DPO surpasses the previous SOTA methods (DMT) by 5.65%, 1.02%, 10.52%, and 3.90% for these four vulnerability types. For machine unauditable vulnerabilities, Smart-LLaMA-DPO surpasses the previous best baseline (iAudit) with an increase of 25.98% in F1 score and 18.25% in accuracy. Both LLM evaluation and human evaluation show that Smart-LLaMA-DPO produces more accurate, comprehensive, and concise explanations than baselines. Human evaluation gave positive scores (4 or 3 points) in 81.15%, 83.88%, and 94.63% for correctness, thoroughness, and clarity.

The main contributions of this paper are as follows:

- To the best of our knowledge, we are the first to introduce preference-based optimization in smart contract vulnerability detection and explanation.
- We propose Smart-LLaMA-DPO, a new method combining continual pre-training, supervised fine-tuning and direct preference optimization for smart contract vulnerability detection, achieving state-of-the-art performance across four major vulnerability types and machine unauditable vulnerabilities (seven types).
- We validate the effectiveness of Smart-LLaMA-DPO in generating high-quality explanations through both LLM evaluation and human evaluation.

2 Background and Motivation

2.1 Key Terms

Solidity [6] is the primary programming language for Ethereum smart contracts. **Call.value()** is a low-level function for sending Ether that may introduce reentrancy vulnerabilities [17]. **Delegatecall** executes target contract in the caller's context, useful for upgradeable contracts but potentially dangerous [18]. **Block.timestamp** is the block's timestamp that may be manipulated by miners [16]. **SmartBugs dataset [20]** is a curated dataset of vulnerable Ethereum smart contracts used for security research. **Qwen2.5-72B-Instruct [47]** developed by Alibaba and **Mistral-Large-Instruct-2407-123B [36]** developed by Mistral AI are two powerful instruction-tuned LLMs.

2.2 Problem Statement

We propose an automated approach to detect vulnerabilities in smart contracts and provide explanations. Our method assigns a label $\hat{y}$ to each independent smart contract, where $\hat{y} = 1$ indicates the presence of a vulnerability and $\hat{y} = 0$ denotes security. We adopts the smart contract vulnerability taxonomy widely used in several recent studies [46, 70, 74], focusing on four key vulnerability types and seven machine-unauditable vulnerabilities.

Reentrancy vulnerability (RE) occurs when a contract calls an external contract or transfers assets (Ether or tokens) before completing internal state changes, allowing attackers to repeatedly call the vulnerable function and potentially withdraw funds multiple times.

Timestamp dependence vulnerability (TD) occurs when smart contracts rely on block timestamps for critical operations. Miners can manipulate these timestamps, potentially compromising contract integrity and leading to financial losses. This vulnerability often affects contracts using timestamps for random number generation or key decision-making processes.

Integer Overflow/Underflow (IO) occurs when the result of an arithmetic operation exceeds the storage range of the variable. In an overflow, the value "wraps around" to the minimum value for that type, while in an underflow, it "wraps around" to the maximum value. This can lead to unexpected contract behavior such as incorrect balances or out-of-control loops.

Delegatecall (DE) is a low-level function call that allows a contract to dynamically load code from another contract. While this provides powerful upgradeability, it can lead to severe security vulnerabilities if used improperly. The main risk is that the called contract executes in the context of the calling contract and can thus modify the calling contract's storage.

Machine-unauditable Vulnerabilities (MU) represent vulnerabilities that are difficult to detect through automated tools and require domain expertise to identify. These include: **Price Oracle Manipulation** (PO) from improper use of price oracle APIs, **Erroneous Accounting** (EA) from incorrect business model calculations, **ID Uniqueness Violations** (IU) from failures in ensuring unique identifiers, **Inconsistent State Updates** (IS) from incorrect correlated state variable updates, **Privilege Escalation** (PE) from insufficient access control, **Atomicity Violations** (AV) from interfering concurrent flows, and **Contract Implementation Specific** (CI) vulnerabilities requiring implementation context understanding.

We focus on these four vulnerability types and machine-unauditable vulnerabilities for the following reasons: (i) These four vulnerabilities account for approximately 70% of financial losses and occur with higher frequency in Ethereum smart contracts [10, 21, 43]. (ii) According to OWASP 2023 Smart Contract Top 10 [42], three of them are ranked as the top-3 vulnerabilities, with delegatecall also included as unchecked external calls. (iii) Recent research [74] shows machine-unauditable vulnerabilities represent a significant security challenge, with price oracle manipulation causing at least $44.8M in losses (34% of real-world exploits).

2.3 Motivations

In this section, we analyze the motivations for improving smart contract vulnerability detection. Using real-world examples and datasets, we illustrate the limitations of current methods, emphasizing the need for high-quality datasets and DPO training.

Table 1. Comparison of Smart Contract Vulnerability Datasets.

Dataset	Label	Explanation	Location	DPO Format	Types
Dataset A [68]	✓	✗	✗	✗	1
Dataset B [76]	✓	✗	✗	✗	2
Dataset C [71]	✓	✗	✗	✗	3
Dataset D [46]	✓	✗	✗	✗	4
iAudit [34]	✓	✓	✓	✗	2*
Our Dataset	✓	✓	✓	✓	5+**

2* : iAudit focuses on logic and traditional vulnerabilities

5+** : includes 4 basic types and machine unauditable vulnerabilities (7 types) [74]

Motivation 1: Insufficient Quality and Comprehensiveness of Datasets. We conducted a systematic evaluation of existing smart contract vulnerability datasets. As shown in Table 1, while Datasets A [68], B [76], C [71], and D [46] provide vulnerability labels, they lack detailed vulnerability explanations and specific location information. Regarding vulnerability types, these datasets either have limited coverage, or like iAudit, focus mainly on logic vulnerabilities without detailed classification of traditional vulnerabilities. Notably, iAudit effectively leverages LLMs to provide vulnerability explanations and location information. While the positive samples are from verified audit reports, the negative samples and explanation enhancement rely on LLMs (including GPT-4 and GPT-3.5). Although constraints were added during explanation enhancement to ensure alignment with actual vulnerabilities, the process could benefit from more systematic verification

by human experts. In contrast, our dataset not only utilizes multiple LLMs (Qwen2.5 and Mistral) for initial explanation generation but also introduces a rigorous expert review process. More importantly, by constructing expert-verified pairs of high-quality and suboptimal explanations (for example, for a reentrancy vulnerability, high-quality explanations detail the trigger conditions, attack paths, and specific impacts, while suboptimal explanations merely mention the presence of reentrancy risk or misunderstand critical state update sequences), our dataset can support large language models in DPO training, which is crucial for generating explanations aligned with human expert preferences. Our dataset not only comprehensively covers four major vulnerability types (RE, TD, IO, and DE) but also includes 7 types of MU vulnerabilities identified in recent research [74].

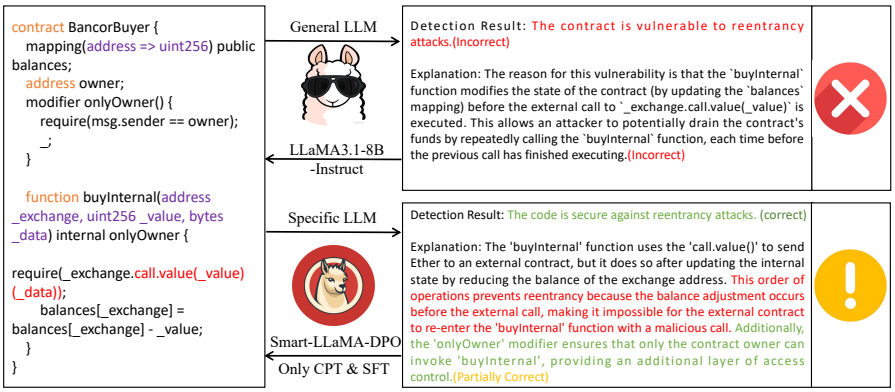

Fig. 1. An motivation example to illustrate the limitations of LLM-based Explainable Smart Contract Vulnerability Detection, motivating the need for Direct Preference Optimization (DPO).

Motivation 2: Limitations in Detection Performance and Explanation Quality of LLM-based Smart Contract Vulnerability Detection Methods. As shown in Fig. 1, general LLMs (LLaMA3.1-8B-Instruct) fail in two aspects: they incorrectly identify a non-existent reentrancy vulnerability and provide inaccurate explanations. Although specialized LLMs like Smart-LLaMA-DPO (only trained through CPT and SFT) perform better in vulnerability detection, they still struggle to provide fully accurate explanations. The specialized LLM correctly identifies that the contract is safe against reentrancy attacks. However, its explanation contains a critical misunderstanding of the operation order. It erroneously states that the balance reduction occurs before the external call, whereas in the original code, the external call is executed first. This limitation in explanation quality persists despite the improvements from CPT and SFT. It underscores the need for further refinement in the LLM's understanding of smart contract vulnerability.

Unlike traditional training methods that rely on absolute labels, DPO learns from relative preferences between paired explanations, which is particularly crucial for smart contract security analysis. This approach is especially effective in two key aspects:

(1) **Fine-grained Security Pattern Learning:** DPO's loss function explicitly models the relative preference between two explanations, enabling the capture of subtle security-critical details in smart contracts. For instance, in reentrancy vulnerability detection, it can learn the critical distinction between explanations that correctly identify the "checks-effects-interactions" pattern versus those that misunderstand execution order. This is crucial as minor execution order misunderstandings can cause catastrophic security breaches in smart contracts.

(2) **Context-dependent Quality Assessment:** Smart contracts often contain complex interactions between multiple functions and state variables, making the quality of explanations

highly context-dependent. DPO's preference learning mechanism is particularly effective here because it can capture how explanation quality varies with context through paired comparisons. For example, consider two explanations for a function containing both external calls and state updates:
Explanation A: "This function updates the balance before making external calls" Explanation B: "This function makes external calls after completing all internal state changes, following the checks-effects-interactions pattern"
While both explanations describe the same code behavior, Explanation B would be preferred in a reentrancy vulnerability analysis context as it explicitly links the ordering to a security pattern. However, for analyzing integer overflow vulnerabilities in the same function, Explanation A's focus on balance updates might be more relevant.

Through this preference learning mechanism, DPO effectively addresses the limitations observed in models trained only with CPT and SFT. For instance, it helps prevent misunderstandings about operation ordering in reentrancy detection which is shown in Fig. 1.

3 Approach

Our Smart-LLaMA-DPO approach consists of four key stages, as illustrated in Fig. 2: SFT and DPO Data Construct, Continual Pre-Training, Supervised Fine-Tuning, and Direct Preference Optimization. The process is supported by initial open-source smart contract vulnerability data collection and final evaluation of explanations. We chose LLaMA-3.1-8B as the base model for its open-source nature, efficient parameter size, strong fine-tuning potential, and proven effectiveness [4]. A comparative evaluation with Qwen2.5-7B and CodeLLaMA-7B using 40 samples from our evaluation dataset demonstrated that LLaMA-3.1-8B provides the most stable performance.

3.1 Open-source Smart Contract Vulnerability Data Collection

3.1.1 Continual Pre-training. We based our Continual Pre-training dataset on research from [52], which underwent extensive filtering and quality checks. The process involved using Google BigQuery to identify all Ethereum blockchain smart contract addresses with at least one transaction. The researchers then accessed Etherscan to obtain the corresponding source code.

To ensure uniqueness, we employed the Jaccard Index [3] for token-based similarity detection. Smart contracts were decomposed into core business logic, library code, and imported files. This approach removed commonly repeated code while preserving unique implementation logic. Following [52, 62], we grouped contracts by filename and filtered them using a 0.9 similarity threshold, eliminating contracts with over 90% token similarity.

3.1.2 Supervised Fine-Tuning and Direct Preference Optimization. For Supervised Fine-Tuning and Direct Preference Optimization, we utilized labeled datasets from [31], [71] and [74], covering various smart contract vulnerability types. The smart contract in [31] were manually verified for accuracy. [71] is based on the SmartBugs dataset [20], with reentrancy vulnerabilities annotated by [68]. Zhang et al. [74] collects machine unauditable vulnerabilities from the highly reputable Code4rena contests. We selected 1,634 smart contracts containing `call.value` from [71] and identified 379 contracts with `delegatecall`, which we manually labeled. To expand our dataset, we collected verified contracts from Etherscan, GitHub repositories, and blog posts, all dated 2020-2024.

3.2 SFT and DPO Data Construct

3.2.1 Initial Data Generation. We designed a multi-stage data generation process for our study, built upon recent work [63, 70]. We utilized Qwen2.5-72B-Instruct and Mistral-Large-Instruct-2407-123B as our initial models to generate detailed explanations of smart contract vulnerabilities.

These open-source LLMs were chosen for their powerful natural language processing and code comprehension capabilities, comparable to GPT-4 Turbo but at a lower cost [36, 47, 53, 69].

We designed specialized prompts for each vulnerability type (RE, TD, DE, IO, and MU), implementing label-guided analysis to help the LLM explain vulnerabilities and pinpoint their exact locations in the code.

For each vulnerability type, the prompts were tailored to focus on specific aspects:

- RE: Use of call.value(), operation order, external calls, access control, and internal function implementation.
- TD: Use of block.timestamp or now, time constraints in critical operations, potential miner manipulation, and precision of time measurements impacting contract logic.
- DE: Use of delegatecall(), context preservation, state variable manipulation, access control, and internal function implementation.
- IO: Arithmetic operations (especially on uint variables), use of SafeMath library or Solidity 0.8.x built-in overflow/underflow checks, use of the 'unchecked' keyword (Solidity 0.8.x or higher), arithmetic operations in critical operations, and type conversion.
- MU: Based on audit report analysis, prompts guide the LLM to identify vulnerabilities identified in [74], explaining their root causes and exploit mechanisms in detail.

3.2.2 LLM Scoring. We employed Llama-3.1-70B-Instruct (validated in Section 3.6.2) as an evaluation model to assess explanations from Qwen2.5 and Mistral based on three criteria: correctness (0.6 weight), thoroughness (0.3 weight), and clarity (0.1 weight), each on a 1-10 scale. Correctness evaluates vulnerability identification accuracy and reasoning logic. Thoroughness assesses coverage of vulnerability issues. Clarity measures explanation structure and applicability. Explanations with the highest weighted composite scores (WCS = 0.6×correctness + 0.3×thoroughness + 0.1×clarity) were selected for human review.

3.2.3 SFT Data Construction. To minimize potential bias, we selected 8 senior PhD students specializing in blockchain: 4 with extensive smart contract security auditing experience and 4 with academic publications in top conferences. They were divided into 4 balanced groups, each containing one industry expert and one academic expert to review one specific vulnerability type. To ensure consistent vulnerability labeling, we established specific annotation guidelines: For **RE**, our labeling criteria cover execution order analysis between external calls and state variable updates, reentrancy risk assessment standards in cross-contract invocation scenarios, identification of hidden reentrant call points in contract inheritance (both ETH and token transfers), and evaluation of reentrancy guard effectiveness. For **TD**, we examine block.timestamp's indirect impacts on contract state transitions. For randomness-related scenarios, we evaluate specific risks of timestamp-based random number generation in applications like gaming and lottery systems. For **IO**, we consider analysis of multiple overflow possibilities in complex expressions and overflow risks from type conversions (e.g., uint8 to uint256). The annotation covers implementation requirements for overflow checking mechanisms across different Solidity versions and potential issues from improper 'SafeMath' usage. For **DE**, we refine storage layout analysis standards in proxy patterns. For contract upgrade mechanisms, we include access control defect identification criteria, verification methods for callee contract code integrity, and security risk assessment standards across different upgrade scenarios. For **MU**, we verify whether the LLM-generated explanations align with the corresponding audit reports. Any discrepancies are corrected to strictly match the reports. Experts reviewed high-scoring LLM explanations and made necessary corrections. To ensure review consistency, when significant disagreements arise between the two reviewers within a group, a third expert from another group is selected to arbitrate. The final modifications are

determined through a three-way discussion. To address potential bias, we introduced an external review stage after the initial annotation. Two senior blockchain and smart contract developers with extensive experience reviewed the annotated explanations which ensured broader perspectives.

3.2.4 DPO Data Construction. The DPO dataset consists of pairs of preferred and rejected outputs. Preferred outputs are taken from the high-quality outputs in the SFT dataset, following the annotation guidelines detailed in the SFT phase. For rejected outputs, experts rewrite a "suboptimal" version based on the lower-scoring outputs from LLaMA3.1. These suboptimal versions maintain basic correctness while intentionally reducing analysis depth: for **RE**, they only identify obvious external calls without analyzing execution ordering, inheritance-based reentrant risks, or token transfer scenarios; for **TD**, they merely mark direct block.timestamp usage without examining state transition impacts or analyzing timestamp-based randomness in gaming/lottery applications; for **IO**, they only flag simple overflow points without considering complex expressions, type conversions, or version-specific checking requirements; for **DE**, they simply identify basic DELEGATECALL usage without analyzing storage layout in proxy patterns or upgrade mechanism security. For **MU**, rejected outputs simplify or omit key details, such as audit findings or critical reasons. Experts ensure a noticeable but reasonable quality gap between preferred and rejected outputs.

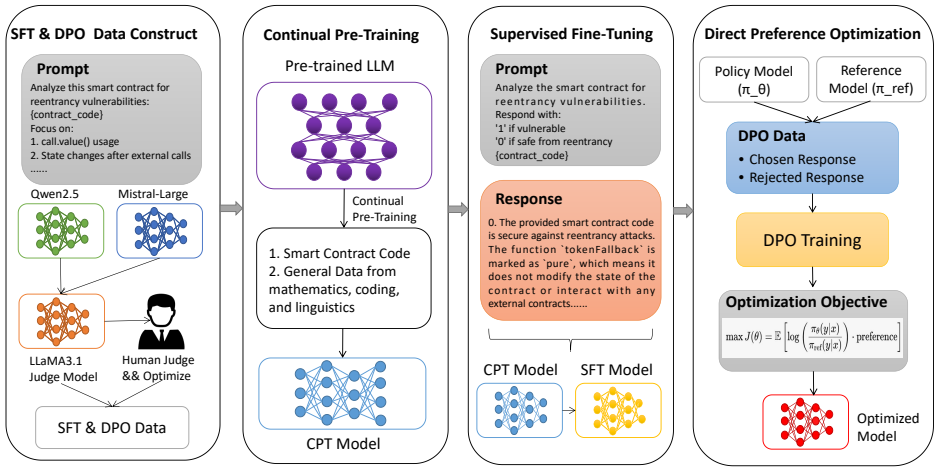

Fig. 2. The Overview of our Smart-LLaMA-DPO.

3.3 Continual Pre-Training

In the Continual Pre-Training phase, we focus on enhancing the LLM's comprehension of smart contract security. This process is guided by a domain-specific language modeling objective:

$$L_{CPT} = -\mathbb{E}x \sim D[\sum i = 1^n \log P(x_i|x_{<i}, c_i)] \tag{1}$$

Here, D represents our curated dataset of smart contracts, x is a sequence of tokens from a contract, c_i denotes the contextual information surrounding x_i, and $P(x_i|x_{<i}, c_i)$ is the LLM's predicted probability of token x_i given its preceding tokens and context.

This formulation encourages the LLM to learn key aspects: (1) Solidity syntax and semantics (keywords like 'function', 'uint256', 'require') for contract comprehension. (2) Design patterns

like 'Checks-Effects-Interactions' to prevent reentrancy attacks, along with other security best practices, helping the LLM identify vulnerabilities. (3) Security-critical functions ('transfer', 'send', 'call.value()'), crucial for fund transfers and inter-contract interactions, allowing the LLM to recognize their correct usage and risks. (4) Contextual analysis skills to detect vulnerabilities by understanding contract structures, function interactions, and state variables. To maintain general capabilities and avoid catastrophic forgetting, diverse data from mathematics, coding, and linguistics is incorporated, enhancing its ability to generalize to novel contract patterns.

3.4 Supervised Fine-Tuning

The Supervised Fine-Tuning phase focuses on optimizing the LLM for vulnerability detection and explanation generation. We employ a balanced objective function:

$$L_{SFT} = \frac{1}{2}(L_{detect} + L_{explain}) \tag{2}$$

where:

$$L_{detect} = -\sum_{(x,y)\in D_{vul}} \log P(y|x;\theta) \tag{3}$$

$$L_{explain} = -\sum_{(x,e)\in D_{exp}} \sum_{i=1}^{|e|} \log P(e_i|x, e_{<i};\theta) \tag{4}$$

In this formulation, x denotes the input smart contract code, while y and e represent the target outputs for the detection task (vulnerability labels) and generation task (explanations), respectively. The parameter θ encompasses all trainable components of the LLM, including attention mechanisms, feedforward networks, and embedding layers.

This balanced loss function ensures that the LLM is effectively trained on both vulnerability detection and explanation generation tasks. The domain-specific knowledge acquired during the Continual Pre-Training stage serves as a crucial foundation for this fine-tuning phase. It enables the LLM to comprehend the intricate context and subtle nuances of smart contracts while simultaneously honing its ability to detect vulnerabilities and generate informative explanations.

3.5 Direct Preference Optimization

In the final stage, we employ Direct Preference Optimization (DPO) [48] to align the LLM's outputs with the preferences of human experts in smart contract vulnerability analysis. DPO offers a simplified approach to preference learning without explicit reward modeling or reinforcement learning.

The core of DPO is based on the insight that the optimal policy π^* for a reward function r^* under a KL-constrained optimization objective can be expressed as:

$$\pi^*(y|x) = \frac{1}{Z(x)}\pi_{\text{ref}}(y|x)\exp\left(\frac{1}{\beta}r^*(x,y)\right) \tag{5}$$

where $Z(x)$ is a normalization factor (also known as the partition function), π_{ref} is a reference policy, and β is a temperature parameter. In our smart contract vulnerability detection scenario:

- x: represents the input smart contract code snippet
- y: represents the LLM-generated vulnerability analysis and explanation
- π_{ref}: is our reference policy, typically the initially fine-tuned model such as the Smart-LLaMA-DPO model trained in the SFT stage
- π^*: is the final optimized model we aim to obtain, capable of generating vulnerability analyses that align with expert preferences

By rearranging this equation, we can express the reward function in terms of the optimal policy:

$$r^*(x, y) = \beta \log \frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)} + \beta \log Z(x) \tag{6}$$

This allows us to reformulate the Bradley-Terry preference model in terms of policies rather than rewards:

$$p^*(y_1 \succ y_2|x) = \sigma\left(\beta \log \frac{\pi^*(y_1|x)}{\pi_{\text{ref}}(y_1|x)} - \beta \log \frac{\pi^*(y_2|x)}{\pi_{\text{ref}}(y_2|x)}\right) \tag{7}$$

where σ is the logistic function, and y_1 and y_2 represent two different vulnerability analysis results.

Based on this, we can define the DPO loss function:

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x, y_w, y_l)\sim\mathcal{D}}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\right)\right] \tag{8}$$

where:

- (x, y_w, y_l) are triples from our curated dataset $\mathcal{D}$, representing input, preferred output, and non-preferred output respectively
- x: smart contract code snippet that serves as the input for vulnerability analysis and forms the context for the LLM's assessment
- y_w: expert-preferred vulnerability analysis result, which represents the high-quality output that aligns with expert judgment and security best practices
- y_l: expert non-preferred vulnerability analysis result, typically containing errors, omissions, or less comprehensive explanations compared to y_w
- π_θ: our policy model being optimized, initialized as π_{ref}, which learns to generate outputs that align with expert preferences through the training process

This loss function encourages the LLM to assign higher probability to preferred outputs (y_w) compared to non-preferred outputs (y_l), effectively learning from expert preferences. By minimizing this loss, we directly optimize our policy model π_θ to align with the preferences of smart contract security experts, without the need for a separate reward model.

3.6 Evaluation of Explanations

To comprehensively evaluate the quality of vulnerability explanations, we developed an evaluation framework inspired by [61, 70], focusing on three aspects: correctness, thoroughness, and clarity. Each aspect is rated using a 4-point Likert scale [26]. It differs from the 10-point scoring system used in dataset annotation (Section 3.2.2). The 10-point system offers fine granularity for curating high-quality explanations, while the 4-point Likert scale emphasizes simplicity and practicality.

3.6.1 Evaluation Criteria. Our evaluation focuses on three core aspects using a 4-point scale (1-Strongly disagree to 4-Strongly agree):

Correctness: Measures logical reasoning and vulnerability identification accuracy. Scores range from 1 (significant errors in logic and identification) to 4 (correct logic with precise identification and localization).

Thoroughness: Assesses comprehensive coverage of all potential vulnerability points. Scores range from 1 (omits key vulnerabilities) to 4 (comprehensive coverage with detailed explanations).

Clarity: Evaluates whether explanations are clear, concise, and applicable. Scores range from 1 (verbose with unclear key points) to 4 (precise, lucid, directly applicable, no redundancy).

3.6.2 LLM Evaluation. We employed Llama-3.1-70B-Instruct for automated assessment, selected after comparing multiple LLMs on 40 explanation samples (8 per vulnerability type). Llama-3.1-70B-Instruct achieved 90% agreement with human expert judgments, matching GPT-4 Turbo and outperforming GPT-4o (87.5%), Qwen2.5-72B-Instruct (85%), and GPT-3.5-Turbo (82.5%), while being more cost-effective. Using carefully crafted prompts with scoring guidelines, the LLM evaluated explanations on correctness, thoroughness, and clarity using a 4-point Likert scale [26].

3.6.3 Human Evaluation. To further validate the evaluation results, we enlisted five experienced smart contract security experts for human evaluation. Each expert dedicated 8 hours to assessing explanations for one vulnerability category, totaling 40 hours. The experts utilized the same 4-point Likert scale [26] as the LLM, scoring each explanation on correctness, thoroughness, and clarity. They also provided detailed scoring rationales and overall quality assessments.

4 Experiments

4.1 Research Questions

To evaluate our Smart-LLaMA-DPO approach, we examine the following research questions:

•**RQ1**: How does Smart-LLaMA-DPO perform in detecting four key smart contract vulnerabilities compared to state-of-the-art methods?

•**RQ2**: Can Smart-LLaMA-DPO effectively detect machine-unauditable vulnerabilities that traditionally require manual analysis?

•**RQ3**: How do individual modules and Chain of Thought (COT) reasoning affect Smart-LLaMA-DPO's effectiveness?

•**RQ4**: How effective are the explanations generated by Smart-LLaMA-DPO in terms of correctness, thoroughness, and clarity?

•**RQ5**: How does Smart-LLaMA-DPO compare to existing methods in terms of practical applicability in real-world scenarios?

4.2 Dataset

Continual Pre-training: We employ a dataset derived from the work of Storhaug et al. [52]. This dataset comprises 186,397 unique smart contract instances from the Ethereum blockchain, totalling 501.62M tokens. We also augment this dataset with an additional 100,000 instances from various domains, including general code, mathematics, English, and Chinese text, totalling 118.94M tokens. This results in a comprehensive dataset of 286,397 instances, totaling 620.56M tokens.

Supervised Fine-Tuning: Our Supervised Fine-Tuning dataset is curated from multiple sources, primarily drawing from Liu et al. [31] and Yu et al. [71]. We incorporate manually verified vulnerabilities from [31] and the SmartBugs dataset [20], with RE vulnerabilities annotated by Wu et al. [68]. Additionally, the dataset includes MU vulnerabilities extracted from Zhang et al. [74]. To enrich the dataset, we manually annotated contracts from [71], Etherscan, GitHub repositories, and blog posts, all dated 2020-2024. For Etherscan-sourced contracts specifically, we focused on token-level RE vulnerabilities. All datasets for SFT and DPO adhere to strict selection criteria: they are publicly available, peer-reviewed, and verified real-world contracts. The final dataset comprises 3,390 RE, 1,167 TD, 1,013 IO, 698 delegatecall, and 1281 MU instances, totalling 8.90M tokens.

Direct Preference Optimization: Our DPO dataset draws from the same sources as our SFT dataset, with no overlap between the two. The dataset contains 270 RE, 227 TD, 260 IO, 265 DE, and 420 MU instances, totaling 1.98M tokens.

Evaluation: Our evaluation dataset combines four vulnerability types from [46] (RE, TD, IO, DE) and machine-unauditable (MU) vulnerabilities from [74]. Following [46]'s methodology, we sampled 20% for RE, TD, and IO, while including all DE samples due to their limited number. We enhanced

the dataset with ERC20 token RE cases and collected additional samples for all vulnerability types from EtherScan, GitHub repositories, and blog posts, all dated 2020-2024. We performed systematic cleaning to remove identical duplicates and incorrect labels (e.g., cases mistakenly labeled as vulnerable where there were no state changes after call.value). The refined dataset contains 3,542 samples: RE (116/470, 13.27%), TD (672/896, 25.30%), IO (354/1,458, 41.16%), DE (76/340, 9.60%), and MU (116/378, 10.67%). MU vulnerabilities include seven categories: PO (25/57), PE (15/46), IU (13/53), IS (23/59), EA (20/55), CI (10/55), and AV (10/53). Numbers in parentheses represent (vulnerable/total samples, percentage in evaluation dataset). The MU vulnerabilities are sourced from Zhang et al. [74], EtherScan, GitHub repositories, and blog posts, with no overlap between evaluation data and training data. For explanation quality evaluation, considering manual review costs, we maintained the same vulnerable/total samples ratios as in the refined dataset and randomly sampled approximately 200 samples per vulnerability type, resulting in 1,061 samples: RE (58/235), TD (142/224), IO (59/243), DE (38/170), and MU (58/189).

4.3 Baselines

Our evaluation includes a range of baselines for Smart Contract Vulnerability Detection, representing state-of-the-art approaches in four categories: rule-based, neural network-based, pre-trained model-based, and LLM-based techniques. **Rule-based techniques** include tools such as Mythril [37], Osiris [58], Oyente [33], Slither [19], Conkas[60], Smartian [11], Confuzzius [57], sFuzz [39], Solhint [44], Sailfish [7], Securify [59], and Smartcheck [56]. **Neural network-based techniques** include GCN [28], TMP [76], AME [30], SMS [46] and DMT [46]. We faced challenges reproducing some neural network-based baselines, and in cases of significant discrepancies, we used the higher of the reproduced or reported results for fairness in comparisons. **Pre-trained models-based techniques**, rely on pre-trained models like CodeT5 [64] and fine-tuning techniques to identify smart contract vulnerabilities, including Peculiar [68] and PSCVFinder [71]. **LLM-based techniques**, rely on LLMs to identify smart contract vulnerabilities, including Llama3.1-8B-Instruct [15], Llama3.1-70B-Instruct [15], Qwen2.5-7B-Instruct [69], Qwen2.5-72B-Instruct [69], GPT-4o (gpt-4o-2024-08-06) [41], Claude-3.5-Sonnet (claude-3-5-sonnet-20241022) [5], GPTScan [54], GPTLens [25], and fine-tuning approaches such as FTSmartAudit [66] and iAudit [34]. All baselines were evaluated under consistent conditions using the same evaluation dataset. To ensure fairness, we fine-tuned FTSmartAudit [66] and iAudit [34] on their original datasets using their respective configurations but observed performance declines on our broader evaluation dataset. To address this, we fine-tuned on our training dataset to adapt them to the expanded evaluation scope.

4.4 Metrics

We evaluated our model performance from two dimensions: vulnerability identification capability and explanation quality. For vulnerability identification, we adopted four standard metrics: Precision measures the ratio of actual vulnerabilities among predicted positives, Recall indicates the proportion of detected vulnerabilities among all actual vulnerabilities, F1-score provides a balanced measure through the harmonic mean of Precision and Recall, and Accuracy reflects the overall correctness of predictions. For vulnerability explanation quality, we employed three key metrics: Correctness, Thoroughness, and Clarity, with detailed evaluation criteria specified in Section 3.6.1.

4.5 Implementation Details

We perform Continual Pre-training, Supervised Fine-Tuning and Direct Preference Optimization using LlamaFactory [75] and DeepSpeed [49] with `fp16` enabled. We calculate loss with cross-entropy and optimize parameters using AdamW [32] with β=(0.9, 0.99) and ϵ=1e-8. For all our models, we employ full parameter tuning. During Continual Pre-training, we set the batch size to 64

Table 2. The performance of our method compared with baselines in terms of Accuracy, Precision, Recall and F1-score (Part 1). Note: LLM-based techniques use Instruct versions of Llama3.1 and Qwen2.5 models.

Methods	Reentrancy				Timestamp Dependency			
	A(%)	P(%)	R(%)	F1(%)	A(%)	P(%)	R(%)	F1(%)
Mythril	54.21	33.75	73.97	46.35	43.27	44.63	44.13	44.38
Osiris	30.77	26.80	91.78	41.49	51.03	52.50	36.63	43.15
Oyente	68.50	42.35	49.32	45.57	53.98	66.67	18.60	29.09
Slither	37.73	16.08	31.51	21.30	57.02	56.50	70.39	62.69
Smartcheck	43.22	30.10	84.93	44.44	51.00	57.14	17.88	27.23
Conkas	68.09	38.36	48.28	42.75	17.86	32.79	28.17	30.30
Smartian	53.62	27.03	51.72	35.50	23.66	39.26	37.32	38.37
Confuzzius	74.04	48.05	63.79	54.81	–	–	–	–
sFuzz	48.09	10.98	15.52	12.86	16.96	31.67	26.76	29.01
Solhint	65.53	36.78	55.17	44.14	35.27	48.60	36.62	41.77
Sailfish	76.17	51.16	75.86	61.11	–	–	–	–
Securify	55.32	28.04	51.72	36.36	–	–	–	–
GCN	73.21	74.47	73.18	73.82	75.91	74.93	77.55	76.22
TMP	76.45	76.04	75.30	75.67	78.84	78.68	76.09	77.36
AME	81.06	79.62	78.45	79.03	82.25	81.42	80.26	80.84
SMS	83.85	79.46	77.48	78.46	89.77	89.15	91.09	90.11
DMT	89.42	83.62	81.06	82.32	94.58	93.60	96.39	94.97
Peculiar	57.46	35.16	47.76	40.53	68.30	77.10	71.13	73.96
PSCVFinder	56.56	35.05	50.75	41.45	39.29	52.17	50.70	51.41
LLaMA3.1-8B	33.19	25.13	86.21	38.91	56.25	68.64	57.04	62.31
Qwen2.5-7B	25.53	24.89	100.00	39.86	69.20	68.34	95.77	79.77
LLaMA3.1-70B	24.68	24.68	100.00	39.59	63.39	63.39	100.00	77.60
Qwen2.5-72B	25.96	25.00	100.00	40.00	63.39	63.39	100.00	77.60
GPT-4o	57.45	34.78	82.76	48.98	49.55	80.85	26.76	40.21
Claude-3.5-Sonnet	26.38	25.11	100.00	40.14	70.09	85.05	64.08	73.09
GPTScan	32.77	25.96	93.10	40.60	60.27	62.56	92.96	74.79
GPTLens	27.23	25.33	100.00	40.42	62.50	63.06	98.59	76.92
FTSmartAudit	32.34	26.07	94.83	40.89	71.88	83.19	69.72	75.86
iAudit	72.77	46.88	77.59	58.44	62.50	63.81	94.37	76.14
Smart-LLaMA-DPO	**94.47**	**90.91**	**86.21**	**88.50**	**95.54**	**97.83**	**95.07**	**96.43**

per device, gradient accumulation steps to 16, epochs to 2, learning rate to 1e-5 with cosine decay, warmup steps to 0, cutoff length to 2048, and save steps to 500. During Supervised Fine-Tuning, we set the batch size to 8 per device, gradient accumulation steps to 8, epochs to 3, learning rate to 1e-5 with cosine decay, warmup steps to 0, cutoff length to 2048, and save steps to 50. For DPO training, we set the cutoff length to 1024, batch size to 8 per device, gradient accumulation steps to 1, learning rate to 1e-5 with cosine decay, epochs to 50, and warmup steps to 0. All models were trained on a server equipped with 8 NVIDIA GeForce RTX H800 GPUs, each with 80GB memory. For evaluating our Smart-LLaMA-DPO, we use greedy decoding with do_sample set to `false`.

Table 3. The performance of our method compared with baselines in terms of Accuracy, Precision, Recall and F1-score (Part 2). Note: LLM-based techniques use Instruct versions of Llama3.1 and Qwen2.5 models.

Methods	Overflow/Underflow				Delegatecall			
	A(%)	P(%)	R(%)	F1(%)	A(%)	P(%)	R(%)	F1(%)
Mythril	37.45	25.30	46.67	32.81	60.71	42.99	74.19	54.44
Osiris	61.90	45.33	75.56	56.67	–	–	–	–
Oyente	72.53	60.87	46.67	52.83	64.71	40.00	31.58	35.29
Slither	50.91	32.28	45.56	37.79	52.04	39.04	91.94	54.81
Smartcheck	52.00	31.25	38.89	34.65	54.08	32.93	43.55	37.50
Conkas	51.85	22.12	38.98	28.22	75.29	47.22	89.47	61.82
Smartian	76.95	51.90	69.49	59.42	79.41	52.83	73.68	61.54
Confuzzius	79.84	58.33	59.32	58.82	72.35	41.82	60.53	49.46
sFuzz	79.01	58.33	47.46	52.34	67.06	36.76	65.79	47.17
Solhint	–	–	–	–	69.41	40.54	78.95	53.57
GCN	67.53	69.52	70.93	70.22	65.76	69.01	69.74	69.37
TMP	70.85	70.26	69.47	69.86	69.11	68.18	70.37	69.26
AME	73.24	71.36	71.59	71.47	72.85	70.25	69.40	69.82
SMS	79.36	78.14	72.98	75.47	78.82	76.97	73.69	75.29
DMT	85.64	85.44	74.32	79.49	82.76	84.61	77.93	81.13
Peculiar	74.90	61.64	76.27	68.18	84.12	65.71	60.53	63.01
PSCVFinder	51.03	33.72	49.15	40.00	90.59	76.19	84.21	80.00
LLaMA3.1-8B	54.32	30.00	66.10	41.27	65.29	34.33	60.53	43.81
Qwen2.5-7B	74.90	47.50	32.20	38.38	67.06	40.22	97.37	56.92
LLaMA3.1-70B	79.42	55.42	77.97	64.79	64.12	38.38	100.00	55.47
Qwen2.5-72B	75.55	64.94	55.56	59.88	68.24	41.30	100.00	58.46
GPT-4o	72.02	44.44	61.02	51.43	57.65	34.55	100.00	51.35
Claude-3.5-Sonnet	77.37	51.75	100.00	68.21	54.71	33.04	100.00	49.67
GPTScan	45.68	27.33	74.58	40.00	47.06	29.37	97.37	45.12
GPTLens	57.61	31.67	64.41	42.46	70.59	43.18	100.00	60.32
FTSmartAudit	79.42	59.18	49.15	53.70	56.47	32.69	89.47	47.89
iAudit	69.55	40.96	57.63	47.89	40.59	27.34	100.00	42.94
Smart-LLaMA-DPO	**94.65**	**94.23**	**83.05**	**88.29**	**94.12**	**100.00**	**73.68**	**84.85**

4.6 Experimental Results

1) RQ1: To answer this question, we compared Smart-LLaMA-DPO's performance against baseline methods across four different vulnerability types. The results are presented in Table 2 and Table 3.

Smart-LLaMA-DPO excelled across all vulnerability types, consistently outperforming state-of-the-art (SOTA) methods. For Reentrancy vulnerabilities, it achieved an F1-score of 88.50% and accuracy of 94.47%, surpassing DMT by 7.51% in F1-score and 5.65% in accuracy. In Timestamp Dependency detection, it led with an F1-score of 96.43% and accuracy of 95.54%, outperforming DMT by 1.54% and 1.02%, respectively. For Overflow/Underflow vulnerabilities, it maintained the top position with an F1-score of 88.29% and accuracy of 94.65%, exceeding DMT by 11.07% and 10.52%. Finally, for Delegatecall vulnerabilities, it achieved an F1-score of 84.85% and accuracy of 94.12%, surpassing PSCVFinder by 6.05% in F1-score and 3.90% in accuracy. Compared to iAudit

and FTSmartAudit, Smart-LLaMA-DPO benefits from the integration of DPO and CPT. iAudit's two-stage architecture separates detection and explanation tasks, which may affect its ability to leverage contextual information effectively. Meanwhile, FTSmartAudit, which lacks DPO, cannot achieve fine-grained preference learning. Notably, Smart-LLaMA-DPO consistently outperformed static analysis tools such as Mythril, Slither, Smartian, Solhint and Conkas, as well as LLM-based approaches including GPT-4o, Claude-3.5-Sonnet, GPTScan, and GPTLens across all metrics.

Answer to RQ1: Smart-LLaMA-DPO consistently outperformed state-of-the-art baseline methods across all four types of vulnerabilities (reentrancy, timestamp dependency, integer overflow/underflow, and delegatecall).

2) RQ2: We evaluate Smart-LLaMA-DPO's capability in detecting 7 types of machine-unauditable vulnerabilities identified in [74]. Table 4 presents the comparative results with baseline methods. Smart-LLaMA-DPO demonstrates superior performance across all seven vulnerability types. For Price Oracle Manipulation, it achieves 87.7% accuracy and 86.3% F1-score, significantly surpassing iAudit. Notably, for Privilege Escalation, it achieves 91.3% accuracy and 84.6% F1-score, with improvements of 2.47% and 3.80% compared to iAudit. Even for complex vulnerabilities like Inconsistent State Updates, it maintains robust performance with 89.8% accuracy and 85.0% F1-score.

Table 4. Performance Comparison on Machine-unauditable Vulnerabilities. LLaMA3.1-8B denotes LLaMA3.1-8B-Instruct. Total represents the overall performance across all MU vulnerabilities.

Method	PO Acc/F1	EA Acc/F1	IU Acc/F1	IS Acc/F1	PE Acc/F1	AV Acc/F1	CI Acc/F1	Total Acc/F1
LLaMA3.1-8B	47.4/60.5	43.7/39.2	58.5/52.2	45.8/57.9	34.8/42.3	20.8/30.0	21.8/31.8	39.2/45.8
GPT-4o	49.1/62.3	54.6/59.0	58.5/50.0	69.5/71.0	67.4/66.7	49.1/27.0	58.2/41.0	57.9/56.4
FTSmartAudit	57.9/25.0	69.1/32.0	86.8/63.2	72.9/50.0	84.8/69.6	84.9/33.3	87.3/46.2	77.3/44.9
iAudit	40.4/55.3	52.7/55.2	88.7/72.7	88.1/82.9	89.1/81.5	88.7/62.5	92.7/80.0	76.7/66.2
Ours	87.7/86.3	87.3/80.0	92.5/83.3	89.8/85.0	91.3/84.6	92.5/77.8	94.6/82.4	90.7/83.4

Answer to RQ2: Smart-LLaMA-DPO outperforms all baselines in detecting 7 types of machine-unauditable vulnerabilities, achieving the highest accuracy and F1-score across all vulnerability types.

3) RQ3: Our ablation study elucidates the crucial roles of DPO and CPT. To provide a feedback loop and enhance explanation transparency, we further incorporate Chain-of-Thought (CoT) reasoning in our analysis. Following LLM4Vuln [53], we design our CoT prompt to first instruct LLMs to summarize the functionality of the smart contract, then analyze vulnerabilities by examining relevant security patterns based on the specific vulnerability type, and finally determine whether it is vulnerable with explanations. The base model represents the full performance, "Base+CoT" shows the base model with COT, "w/o dpo" indicates the model without DPO, "w/o cpt" represents the LLM without CPT, and "w/o dpo & cpt" shows the performance without both components.

For RE, removing DPO training significantly reduces performance (accuracy drops from 94.47% to 83.40%, F1 from 88.50% to 73.47%), while removing CPT has a smaller impact (accuracy 86.38%, F1 77.78%). This shows that reentrancy detection relies more on DPO for fine-grained preference learning. This aligns with the need to distinguish subtle differences in the "Checks-Effects-Interactions" pattern, primarily learned through paired examples in DPO training. **For IO**, removing CPT causes a

Table 5. Ablation Study of Smart-LLaMA-DPO. MU represents machine-unauditable vulnerabilities.

Types	Metric	Base	Base+CoT	w/o dpo	w/o cpt	w/o dpo & cpt
RE	Acc(%)	94.47	94.47	83.40	86.38	90.21
	F1(%)	88.50	88.50	73.47	77.78	79.65
TD	Acc(%)	95.54	93.75	80.80	82.14	69.64
	F1(%)	96.43	95.71	85.32	86.39	69.64
IO	Acc(%)	94.65	93.42	89.71	83.95	85.19
	F1(%)	88.29	86.21	81.75	53.01	56.10
DE	Acc(%)	94.12	94.12	93.53	93.53	91.76
	F1(%)	84.85	84.85	83.08	83.08	78.12
MU	Acc(%)	90.74	91.53	78.84	86.24	72.22
	F1(%)	83.41	85.19	71.22	80.60	66.88

significant drop in performance (F1 drops from 96.43% to 53.01%), while removing DPO has minimal impact. Detection relies on understanding Solidity's type system and arithmetic rules, which is mainly acquired through CPT. While DPO improves explanation, core detection relies on CPT's domain knowledge. **For TD**, both CPT and DPO have significant impacts. The full model achieves an F1 score of 96.43%, but removing either component significantly degrades performance, and removing both drops accuracy to 69.64%. This reflects the dual complexity of timestamp dependency vulnerabilities, which require CPT to capture block.timestamp semantics and DPO to optimize risk differentiation in time-dependent scenarios. **For DE**, CPT and DPO have relatively balanced roles. Detection requires CPT to understand EVM storage mechanics and DPO to identify storage layout mismatch risks, with either component's removal leading to moderate performance degradation. **For MU**, results reveal significant reliance on DPO training (removing DPO: accuracy drops from 90.74% to 78.84%, F1 from 83.41% to 71.22%), highlighting the importance of fine-grained preference learning. Removing CPT has a smaller impact (accuracy 86.24%, F1 80.60%), but still shows that domain-specific knowledge from pre-training enhances detection. The integration of CoT reasoning shows mixed effectiveness. For simpler vulnerabilities like TD and IO, performance drops (TD: -1.87% accuracy, -0.75% F1; IO: -1.30% accuracy, -2.36% F1), likely due to CoT overcomplicating these straightforward pattern recognition tasks. For DE and RE vulnerabilities, performance remains stable. However, for complex vulnerabilities like MU, CoT improves performance (+0.87% accuracy, +2.13% F1), highlighting its strength in analyzing complex logic and dependencies.

Answer to RQ3: Our findings confirm that the combination of continual pre-training and DPO training creates strong complementary effects, effectively addressing the diverse requirements of different vulnerability types and achieving SOTA detection performance across all vulnerability types.

4) RQ4: Table 6 presents the quality assessment results of explanations generated by the Smart-LLaMA-DPO and other baselines. The evaluation comprises two parts: LLM Evaluation and Human Evaluation, each assessing three dimensions: Correctness, Thoroughness, and Clarity. Our model significantly outperformed the best baseline (iAudit) in both LLM evaluation and human evaluation. In the LLM evaluation, when combining scores 4 (agree) and 3 (somewhat agree), our model achieved higher total positive ratings across all metrics: 86.62% compared to the baseline's 75.12% for correctness (834+85 and 713+84 respectively), 90.10% compared to 75.12% for thoroughness

(731+225 and 586+211 respectively), and 97.46% compared to 93.03% for clarity (565+469 and 747+240 respectively). Similarly in human evaluation, our model maintained consistently higher combined positive ratings (scores 4 and 3): 81.15% compared to the baseline's 70.22% for correctness (646+215 and 388+357 respectively), 83.88% compared to 72.76% for thoroughness (544+346 and 188+584 respectively), and 94.63% compared to 83.98% for clarity (569+435 and 473+418 respectively).

Answer to RQ4: Smart-LLaMA-DPO has demonstrated its capability to generate explanations for smart contract vulnerability detection that are correct, thorough, and clear, as validated by both LLM Evaluation and Human Evaluation.

Table 6. Ratings of Correctness, Thoroughness, and Clarity (LLaMA3.1-8B refers to the Instruct version).

	Correctness				Thoroughness				Clarity			
	1	2	3	4	1	2	3	4	1	2	3	4
LLM Evaluation												
LLaMA3.1-8B	116	201	165	579	42	229	332	458	30	204	578	249
FTSmartAudit	101	234	161	565	53	167	351	490	41	165	454	401
iAudit	135	129	84	713	48	216	211	586	27	47	240	747
Ours	56	86	85	834	9	96	225	731	2	25	469	565
Human Evaluation												
LLaMA3.1-8B	76	303	326	356	70	266	457	268	29	212	623	197
FTSmartAudit	127	234	352	348	126	184	523	228	28	165	482	386
iAudit	61	255	357	388	48	241	584	188	27	143	418	473
Ours	19	181	215	646	18	153	346	544	9	48	435	569

5) RQ5: We demonstrate the effectiveness of Smart-LLaMA-DPO in smart contract vulnerability detection through a comparative case study as shown in Fig. 3.

Why we choose GPT-4o in GPT family and iAudit for this RQ. GPT-4o (gpt-4o-2024-08-06) was chosen for its superior detection performance, with the highest average accuracy (59.17%) and F1-score (49.70%) across five vulnerability types, outperforming GPT-4 (accuracy: 56.49%, F1-score: 48.50%) and GPT-4 Turbo (accuracy: 50.55%, F1-score: 49.54%). Among works focusing on generating smart contract vulnerability explanations iAudit was selected for its higher average performance (accuracy: 64.43%, F1-score: 58.31%) compared to FTSmartAudit (accuracy: 63.47%, F1-score: 52.64%).

In the FundManager contract, GPT-4o identifies a reentrancy vulnerability risk, noting that the refund function updates the balance after an external call. iAudit also identifies a reentrancy vulnerability but incorrectly assumes that the balance is updated before the external call. In contrast, Smart-LLaMA-DPO provides more nuanced analysis, acknowledging the non-standard pattern usage while explaining how the onlyOwner modifier enhances security. For the LockYourLove contract, while GPT-4o and iAudit warn about potential block.timestamp manipulation by miners, Smart-LLaMA-DPO explains that the timestamp is only used for logging purposes and doesn't affect critical operations or contract logic, thus not constituting an actual vulnerability. For the LendingContract, GPT-4o and Smart-LLaMA-DPO correctly identify the root cause of the Price Oracle Manipulation vulnerability: reliance on the Uniswap pool's instant price. iAudit, however, focuses on secondary issues like access control permissions, reflecting how its two-stage architecture can lead to inconsistencies between detection and explanation.

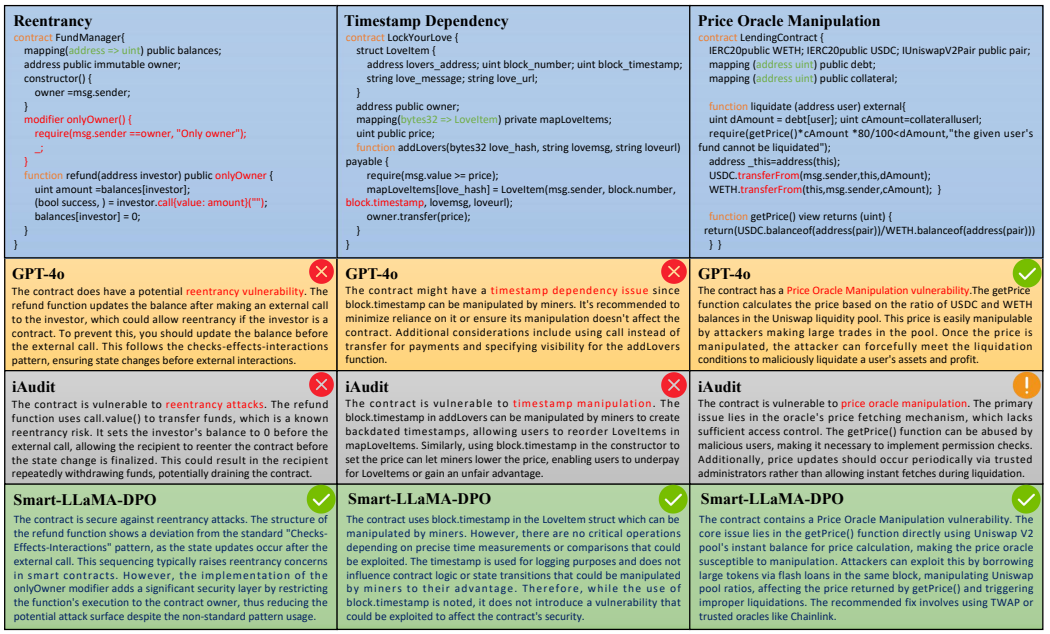

Fig. 3. Case Study of Smart Contract Vulnerability Detection Using Smart-LLaMA-DPO.

Answer to RQ5: Compared to GPT-4o and iAudit which produce false positives, Smart-LLaMA-DPO provides contextual security analysis by accurately assessing both potential risks and implemented security measures, enabling reliable distinction between theoretical vulnerabilities and actual security threats.

5 Related Work

5.1 Smart Contract Vulnerability Detection

Research in smart contract vulnerability detection has evolved significantly. Early approaches relied on program analysis techniques: some used symbolic execution to explore execution paths and identify vulnerabilities [33], while others employed pattern matching against known vulnerability signatures [37, 56] or applied formal verification with custom compliance patterns [59]. The field then progressed to machine learning, particularly deep learning models, offering improved accuracy and generalization. Notable examples include bidirectional LSTMs with attention mechanisms for specific vulnerability detection [45], and deep learning for measuring similarity to known vulnerable contracts [22]. Graph neural networks (GNNs) gained popularity due to their ability to capture contract structural information [30, 76]. Recent advancements include pre-trained models like Peculiar [68] for improved generalization, and innovative approaches such as prompt-tuning to bridge pre-training and downstream tasks [71]. Cross-modality learning approaches have also emerged, leveraging information from both bytecode and source code [46]. The latest frontier involves Large Language Models (LLMs). Studies by Chen et al. [9] and David et al. [13] evaluated LLMs on real-world datasets, revealing both potential and challenges. Hu et al. [25] explored LLMs' reasoning capabilities, while Sun et al. [54] introduced GPTScan for program analysis. FTSmartAudit

[66] fine-tunes smaller LLMs for smart contract auditing, achieving comparable performance to state-of-the-art models. Ma et al. [34] proposed iAudit, a two-stage LLM framework for vulnerability detection and explanation, though its separated design may lead to inconsistencies.

5.2 Reinforcement Learning in Software Engineering

Reinforcement learning (RL) has gained increasing attention in software engineering in recent years [1, 8, 40, 50, 51]. Several studies have explored RL applications in software tasks. Kim et al. [27] introduced ARAT-RL, an adaptive REST API testing technique using RL to prioritize operations. Corradini et al. [12] proposed DeepREST for automated REST API testing using deep RL. In repository-level code completion, Wang et al. [65] presented RLCoder for improving code generation. Li et al. [29] developed IRCoCo, applying deep RL to code completion. More recently, Nashaat et al. [38] proposed the CodeMentor framework, which utilizes reinforcement learning with human feedback (RLHF) to optimize the performance of LLMs in software review tasks. However, the application of these techniques to explainable smart contract vulnerability detection remains unexplored. Our work applies preference-based optimization to this critical task. While not strictly reinforcement learning, our approach shares the goal of improving performance based on feedback.

6 Threats to Validity

Internal Validity: Our Smart-LLaMA-DPO occasionally produces redundant information in its outputs. We currently address this by truncating the generated text to retain only the initial portion. This issue likely stems from the reinforcement of certain patterns during the fine-tuning or DPO training process. To improve output quality, we are exploring advanced post-processing methods and refined post-training techniques for future iterations.

External Validity: The development of Smart-LLaMA-DPO relied heavily on a large corpus of annotated training data. Our data preparation involved a multi-step process using multiple LLMs and human verification. Despite these efforts, we cannot guarantee the absolute accuracy of every reasoning step in our training datasets. Furthermore, our current Smart-LLaMA-DPO's scope is limited to specific vulnerability types due to resource limitations. Future research will focus on developing more efficient data annotation methods and expanding our Smart-LLaMA-DPO's capability to address a wider array of smart contract vulnerabilities.

7 Conclusion

In this paper, we propose Smart-LLaMA-DPO, an advanced smart contract vulnerability detection approach based on LLMs. Unlike prior work, Smart-LLaMA-DPO utilizes a three-stage post-training process: continual pre-training, supervised fine-tuning, and direct preference optimization. In addition, We construct a comprehensive dataset covering four major vulnerability types and machine-unauditable vulnerabilities for both SFT and DPO. Our approach significantly outperforms state-of-the-art methods and can provide reliable explanations.

8 Data Availability

All the experimental data and source code are available at https://doi.org/10.5281/zenodo.15200616

ACKNOWLEDGMENTS

This work was supported by the National Key Research and Development Program of China (No.2023YFB3307203) and the Alliance of International Science Organizations Collaborative Research Program (No.ANSO-CR-KP-2022-03).

References

[1] Amr Abo-eleneen, Ahammed Palliyali, and Cagatay Catal. 2023. The role of Reinforcement Learning in software testing. *Information and Software Technology* (2023), 107325.

[2] Maher Alharby and Aad Van Moorsel. 2017. Blockchain-based smart contracts: A systematic mapping study. *arXiv preprint arXiv:1710.06372* (2017).

[3] Miltiadis Allamanis. 2019. The adverse effects of code duplication in machine learning models of code. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. 143–153.

[4] Kamel Alrashedy and Ahmed Binjahlan. 2023. Language Models are Better Bug Detector Through Code-Pair Classification. *arXiv preprint arXiv:2311.07957* (2023).

[5] Anthropic. 2024. Claude-3.5-Sonnet. https://www.anthropic.com/claude.

[6] The Solidity Authors. 2024. Solidity Documentation. https://docs.soliditylang.org/en/v0.8.28/ Online documentation.

[7] Priyanka Bose, Dipanjan Das, Yanju Chen, Yu Feng, Christopher Kruegel, and Giovanni Vigna. 2022. Sailfish: Vetting smart contract state-inconsistency bugs in seconds. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 161–178.

[8] Partha Chakraborty, Mahmoud Alfadel, and Meiyappan Nagappan. 2024. RLocator: Reinforcement learning for bug localization. *IEEE Transactions on Software Engineering* (2024).

[9] Chong Chen, Jianzhong Su, Jiachi Chen, Yanlin Wang, Tingting Bi, Yanli Wang, Xingwei Lin, Ting Chen, and Zibin Zheng. 2023. When chatgpt meets smart contract vulnerability detection: How far are we? *arXiv preprint arXiv:2309.05520* (2023).

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

[11] Jaeseung Choi, Doyeon Kim, Soomin Kim, Gustavo Grieco, Alex Groce, and Sang Kil Cha. 2021. Smartian: Enhancing smart contract fuzzing with static and dynamic data-flow analyses. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 227–239.

[12] Davide Corradini, Zeno Montolli, Michele Pasqua, and Mariano Ceccato. 2024. DeepREST: Automated Test Case Generation for REST APIs Exploiting Deep Reinforcement Learning. *arXiv preprint arXiv:2408.08594* (2024).

[13] Isaac David, Liyi Zhou, Kaihua Qin, Dawn Song, Lorenzo Cavallaro, and Arthur Gervais. 2023. Do you still need a manual smart contract audit? *arXiv preprint arXiv:2306.12338* (2023).

[14] Vikram Dhillon, David Metcalf, Max Hooper, Vikram Dhillon, David Metcalf, and Max Hooper. 2017. The DAO hacked. *blockchain enabled applications: Understand the blockchain Ecosystem and How to Make it work for you* (2017), 67–78.

[15] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[16] Ethereum Foundation. 2024. Block and Transaction Properties. https://docs.soliditylang.org/en/latest/units-and-global-variables.html#block-and-transaction-properties.

[17] Ethereum Foundation. 2024. Security Considerations - Sending and Receiving Ether. https://docs.soliditylang.org/en/latest/security-considerations.html#sending-and-receiving-ether.

[18] Ethereum Foundation. 2024. Solidity by Example - Delegatecall. https://solidity-by-example.org/delegatecall/.

[19] Josselin Feist, Gustavo Grieco, and Alex Groce. 2019. Slither: a static analysis framework for smart contracts. In *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. IEEE, 8–15.

[20] João F Ferreira, Pedro Cruz, Thomas Durieux, and Rui Abreu. 2020. Smartbugs: A framework to analyze solidity smart contracts. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 1349–1352.

[21] Jianbo Gao, Han Liu, Chao Liu, Qingshan Li, Zhi Guan, and Zhong Chen. 2019. Easyflow: Keep ethereum away from overflow. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 23–26.

[22] Zhipeng Gao, Vinoj Jayasundara, Lingxiao Jiang, Xin Xia, David Lo, and John Grundy. 2019. Smartembed: A tool for clone and bug detection in smart contracts through structural code embedding. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 394–397.

[23] Péter Hegedűs. 2018. Towards analyzing the complexity landscape of solidity based ethereum smart contracts. In *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*. 35–39.

[24] Tharaka Hewa, Mika Ylianttila, and Madhusanka Liyanage. 2021. Survey on blockchain based smart contracts: Applications, opportunities and challenges. *Journal of Network and Computer Applications* 177 (2021), 102857.

[25] Sihao Hu, Tiansheng Huang, Fatih İlhan, Selim Furkan Tekin, and Ling Liu. 2023. Large language model-powered smart contract vulnerability detection: New perspectives. *arXiv preprint arXiv:2310.01152* (2023).

[26] Ankur Joshi, Saket Kale, Satish Chandel, and D Kumar Pal. 2015. Likert scale: Explored and explained. *British journal of applied science & technology* 7, 4 (2015), 396–403.

[27] Myeongsoo Kim, Saurabh Sinha, and Alessandro Orso. 2023. Adaptive rest api testing with reinforcement learning. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 446–458.

[28] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[29] Bolun Li, Zhihong Sun, Tao Huang, Hongyu Zhang, Yao Wan, Ge Li, Zhi Jin, and Chen Lyu. 2024. IRCoCo: Immediate Rewards-Guided Deep Reinforcement Learning for Code Completion. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 182–203.

[30] Zhenguang Liu, Peng Qian, Xiang Wang, Lei Zhu, Qinming He, and Shouling Ji. 2021. Smart contract vulnerability detection: from pure neural network to interpretable graph feature and expert pattern fusion. *arXiv preprint arXiv:2106.09282* (2021).

[31] Zhenguang Liu, Peng Qian, Jiaxu Yang, Lingfeng Liu, Xiaojun Xu, Qinming He, and Xiaosong Zhang. 2023. Rethinking smart contract fuzzing: Fuzzing with invocation ordering and important branch revisiting. *IEEE Transactions on Information Forensics and Security* 18 (2023), 1237–1251.

[32] Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).

[33] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 254–269.

[34] Wei Ma, Daoyuan Wu, Yuqiang Sun, Tianwen Wang, Shangqing Liu, Jian Zhang, Yue Xue, and Yang Liu. 2024. Combining Fine-Tuning and LLM-based Agents for Intuitive Smart Contract Auditing with Justifications. *arXiv preprint arXiv:2403.16073* (2024).

[35] Muhammad Izhar Mehar, Charles Louis Shier, Alana Giambattista, Elgar Gong, Gabrielle Fletcher, Ryan Sanayhie, Henry M Kim, and Marek Laskowski. 2019. Understanding a revolutionary and flawed grand experiment in blockchain: the DAO attack. *Journal of Cases on Information Technology (JCIT)* 21, 1 (2019), 19–32.

[36] Mistral. 2024. Large Enough | Mistral AI | Frontier AI in your hands. https://mistral.ai/news/mistral-large-2407/

[37] B Mueller. 2017. Mythril-Reversing and bug hunting framework for the Ethereum blockchain.

[38] Mona Nashaat and James Miller. 2024. Towards Efficient Fine-tuning of Language Models with Organizational Data for Automated Software Review. *IEEE Transactions on Software Engineering* (2024).

[39] Tai D Nguyen, Long H Pham, Jun Sun, Yun Lin, and Quang Tran Minh. 2020. sfuzz: An efficient adaptive fuzzer for solidity smart contracts. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 778–788.

[40] Amin Nikanjam, Mohammad Mehdi Morovati, Foutse Khomh, and Houssem Ben Braiek. 2022. Faults in deep reinforcement learning programs: a taxonomy and a detection approach. *Automated software engineering* 29, 1 (2022), 8.

[41] OpenAI. 2024. GPT-4-Turbo. https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo.

[42] OWASP Foundation. 2023. OWASP Smart Contract Top 10. https://owasp.org/www-project-smart-contract-top-10/. Accessed: 2024-12-04.

[43] Purathani Praitheeshan, Lei Pan, Jiangshan Yu, Joseph Liu, and Robin Doss. 2019. Security analysis methods on ethereum smart contract vulnerabilities: a survey. *arXiv preprint arXiv:1908.08605* (2019).

[44] Protofire. 2020. Solhint: A linting utility for Solidity code. https://github.com/protofire/solhint. GitHub repository with over 1000 stars, Used by over 54,000 repositories.

[45] Peng Qian, Zhenguang Liu, Qinming He, Roger Zimmermann, and Xun Wang. 2020. Towards automated reentrancy detection for smart contracts based on sequential models. *IEEE Access* 8 (2020), 19685–19695.

[46] Peng Qian, Zhenguang Liu, Yifang Yin, and Qinming He. 2023. Cross-modality mutual learning for enhancing smart contract vulnerability detection on bytecode. In *Proceedings of the ACM Web Conference 2023*. 2220–2229.

[47] Qwen. 2024. Qwen2.5: A Party of Foundation Models. https://qwenlm.github.io/blog/qwen2.5/

[48] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems* 36 (2023), 53728–53741.

[49] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3505–3506.

[50] Sameer Reddy, Caroline Lemieux, Rohan Padhye, and Koushik Sen. 2020. Quickly generating diverse valid test inputs with reinforcement learning. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 1410–1421.

[51] Andrea Romdhana, Alessio Merlo, Mariano Ceccato, and Paolo Tonella. 2022. Deep reinforcement learning for black-box testing of android apps. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 4 (2022), 1–29.

[52] André Storhaug, Jingyue Li, and Tianyuan Hu. 2023. Efficient avoidance of vulnerabilities in auto-completed smart contract code using vulnerability-constrained decoding. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 683–693.

[53] Yuqiang Sun, Daoyuan Wu, Yue Xue, Han Liu, Wei Ma, Lyuye Zhang, Yang Liu, and Yingjiu Li. 2024. Llm4vuln: A unified evaluation framework for decoupling and enhancing llms' vulnerability reasoning. *arXiv preprint arXiv:2401.16185* (2024).

[54] Yuqiang Sun, Daoyuan Wu, Yue Xue, Han Liu, Haijun Wang, Zhengzi Xu, Xiaofei Xie, and Yang Liu. 2024. GPTScan: Detecting Logic Vulnerabilities in Smart Contracts by Combining GPT with Program Analysis. *Proc. IEEE/ACM ICSE* (2024).

[55] Melanie Swan. 2015. *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.".

[56] Sergei Tikhomirov, Ekaterina Voskresenskaya, Ivan Ivanitskiy, Ramil Takhaviev, Evgeny Marchenko, and Yaroslav Alexandrov. 2018. Smartcheck: Static analysis of ethereum smart contracts. In *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*. 9–16.

[57] Christof Ferreira Torres, Antonio Ken Iannillo, Arthur Gervais, and Radu State. 2021. Confuzzius: A data dependency-aware hybrid fuzzer for smart contracts. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 103–119.

[58] Christof Ferreira Torres, Julian Schütte, and Radu State. 2018. Osiris: Hunting for integer bugs in ethereum smart contracts. In *Proceedings of the 34th Annual Computer Security Applications Conference*. 664–676.

[59] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Buenzli, and Martin Vechev. 2018. Securify: Practical security analysis of smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 67–82.

[60] Nuno Veloso. 2021. Conkas. https://github.com/nveloso/conkas. GitHub repository.

[61] Chong Wang, Yiling Lou, Junwei Liu, and Xin Peng. 2023. Generating variable explanations via zero-shot prompt learning. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 748–760.

[62] Pengcheng Wang, Jeffrey Svajlenko, Yanzhao Wu, Yun Xu, and Chanchal K Roy. 2018. CCAligner: a token based large-gap clone detector. In *Proceedings of the 40th International Conference on Software Engineering*. 1066–1077.

[63] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171* (2022).

[64] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 8696–8708.

[65] Yanlin Wang, Yanli Wang, Daya Guo, Jiachi Chen, Ruikai Zhang, Yuchi Ma, and Zibin Zheng. 2024. Rlcoder: Reinforcement learning for repository-level code completion. *arXiv preprint arXiv:2407.19487* (2024).

[66] Zhiyuan Wei, Jing Sun, Zijian Zhang, Xianhao Zhang, and Meng Li. 2024. Leveraging Fine-Tuned Language Models for Efficient and Accurate Smart Contract Auditing. *arXiv preprint arXiv:2410.13918* (2024).

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

[68] Hongjun Wu, Zhuo Zhang, Shangwen Wang, Yan Lei, Bo Lin, Yihao Qin, Haoyu Zhang, and Xiaoguang Mao. 2021. Peculiar: Smart contract vulnerability detection based on crucial data flow graph and pre-training techniques. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 378–389.

[69] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671* (2024).

[70] Lei Yu, Shiqi Chen, Hang Yuan, Peng Wang, Zhirong Huang, Jingyuan Zhang, Chenjie Shen, Fengjun Zhang, Li Yang, and Jiajia Ma. 2024. Smart-LLaMA: Two-Stage Post-Training of Large Language Models for Smart Contract Vulnerability Detection and Explanation. *arXiv preprint arXiv:2411.06221* (2024).

[71] Lei Yu, Junyi Lu, Xianglong Liu, Li Yang, Fengjun Zhang, and Jiajia Ma. 2023. PSCVFinder: A Prompt-Tuning Based Framework for Smart Contract Vulnerability Detection. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 556–567.

[72] Lei Yu, Fengjun Zhang, Jiajia Ma, Li Yang, Yuanzhe Yang, and Wei Jia. 2023. Who Are the Money Launderers? Money Laundering Detection on Blockchain via Mutual Learning-Based Graph Neural Network. In *2023 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[73] Quanjun Zhang, Tongke Zhang, Juan Zhai, Chunrong Fang, Bowen Yu, Weisong Sun, and Zhenyu Chen. 2023. A critical review of large language model on software engineering: An example from chatgpt and automated program repair. *arXiv preprint arXiv:2310.08879* (2023).

[74] Zhuo Zhang, Brian Zhang, Wen Xu, and Zhiqiang Lin. 2023. Demystifying exploitable bugs in smart contracts. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 615–627.

[75] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. *arXiv preprint arXiv:2403.13372* (2024).
[76] Yuan Zhuang, Zhenguang Liu, Peng Qian, Qi Liu, Xiang Wang, and Qinming He. 2020. Smart Contract Vulnerability Detection using Graph Neural Network.. In *IJCAI*. 3283–3290.
[77] Weiqin Zou, David Lo, Pavneet Singh Kochhar, Xuan-Bach Dinh Le, Xin Xia, Yang Feng, Zhenyu Chen, and Baowen Xu. 2019. Smart contract development: Challenges and opportunities. *IEEE Transactions on Software Engineering* 47, 10 (2019), 2084–2106.

Received 2024-10-31; accepted 2025-03-31