УДК 004.75:004.056.55:004.85 DOI https://doi.org/10.35546/kntu2078-4481.2025.3.2.58

О. І. ТЕРЕЩЕНКО
аспірант кафедри інженерії програмного забезпечення
Національний університет «Одеська політехніка»
ORCID: 0000-0003-4510-5255

МЕТОДИ ІН'ЄКЦІЇ ВРАЗЛИВОСТЕЙ У СМАРТ-КОНТРАКТИ ДЛЯ ФОРМУВАННЯ ЗБАЛАНСОВАНИХ ДАТАСЕТІВ

Смарт-контракти широко застосовуються у фінансових та децентралізованих застосунках, проте їхня безпека залишається критичною проблемою. Аналіз існуючих корпусів показує значний дисбаланс: поширені класи уразливостей (integer overflow/underflow) істотно домінують, тоді як критично небезпечна вразливість повторного входу представлена обмежено. Це ускладнює навчання та об'єктивне оцінювання інструментів виявлення вразливостей. Мета дослідження полягає у підвищенні об'єктивності та якості навчання й тестування методів виявлення уразливостей у смарт-контрактах шляхом створення збалансованого контрольованого датасету. Запропоновано два взаємодоповнювальні підходи ін'єкції. Детермінований метод спирається на статичний аналіз і формальні патерни відбору/модифікації функцій, забезпечуючи відтворюваність і синтаксичну коректність. LLM-підхід виконує контекстно узгоджені зміни з мінімальною різницею коду, підвищуючи різноманітність прикладів. Обидва інтегровані в єдиний конвеєр із нормалізацією, дедуплікацією та багатоступеневою валідацією: успішна компіляція solc, статичне підтвердження цільових патернів, перевірка збереження нецільової логіки та мінімальності змін. Результатом є збалансований датасет з п'яти класів (integer_overflow, integer_underflow, timestamp_dependency, reentrancy, «безпечні» контракти) з вирівняною кількістю прикладів, стандартизованим форматом збереження (повний контракт, вразливий фрагмент, метадані) та відтворюваним пайплайном. Поєднання детермінованого й LLM-методів забезпечує баланс контрольованості та реалістичності, що покращує якість експериментів і чесність порівнянь інструментів. Новизна полягає в уніфікованій формальній специфікації операторів ін'єкції та практичному конвеєрі пакетної генерації; обмеження стосуються стохастичності LLM і потреби подальших динамічних PoC-перевірок.

Ключові слова: *смарт-контракти, Solidity, вразливості, ін'єкція вразливостей, reentrancy, integer overflow, збалансований датасет, статичний аналіз, великі мовні моделі.*

O. I. TERESHCHENKO
Postgraduate Student at the Department of Software Engineering
Odesa Polytechnic National University
ORCID: 0000-0003-4510-5255

METHODS OF VULNERABILITY INJECTION INTO SMART CONTRACTS FOR BALANCED DATASET GENERATION

Smart contracts are widely used in financial and decentralized applications; however, their security remains a critical issue. Analysis of existing corpora shows a significant imbalance: common vulnerability classes (integer overflow/underflow) are strongly dominant, while the critically dangerous reentrancy vulnerability is underrepresented. This complicates both training and objective evaluation of vulnerability detection tools. The aim of the study is to improve the objectivity and quality of training and testing methods for detecting vulnerabilities in smart contracts by creating a balanced and controlled dataset. Two complementary injection approaches are proposed. The deterministic method relies on static analysis and formal patterns for selecting and modifying functions, ensuring reproducibility and syntactic correctness. The LLM-based approach performs context-aware modifications with minimal code differences, increasing the diversity of examples. Both approaches are integrated into a unified pipeline with normalization, deduplication, and multi-stage validation: successful solc compilation, static confirmation of target patterns, preservation of non-target logic, and minimization of code changes. The result is a balanced dataset of five classes (integer_overflow, integer_underflow, timestamp_dependency, reentrancy, and "safe" contracts) with an equal number of examples, a standardized storage format (full contract, vulnerable snippet, metadata), and a reproducible pipeline. Combining deterministic and LLM methods provides a balance between controllability and realism, which improves the quality of experiments and the fairness of tool comparisons. The novelty lies in the unified formal specification of injection operators and the practical pipeline for batch dataset generation, while limitations concern the stochastic nature of LLMs and the need for further dynamic PoC validation.

Key words: *smart contracts, Solidity, vulnerabilities, vulnerability injection, reentrancy, integer overflow, balanced dataset, static analysis, large language models.*

Постановка проблеми

Смарт-контракти на платформі Ethereum та інших блокчейн-системах широко застосовуються для фінансових операцій, токенізації активів та реалізації децентралізованих застосунків. Водночас відомо, що навіть незначні уразливості у коді можуть призвести до суттєвих фінансових втрат, як це сталося під час атак на DAO (2016) чи Parity Wallet (2017). Проблема полягає в тому, що існуючі реальні датасети смарт-контрактів містять сильний класовий дисбаланс: одні типи уразливостей (наприклад, integer overflow/underflow) значно переважають, тоді як критично небезпечні, але рідкісні (наприклад, reentrancy) представлені вкрай малою кількістю прикладів. Це унеможливлює ефективне навчання моделей машинного навчання та знижує якість виявлення уразливостей [1, 2].

Таким чином, постає задача створення збалансованого та контрольованого датасету смарт-контрактів, у якому всі цільові класи уразливостей представлені рівномірно. Для цього необхідні методи автоматизованої ін'єкції вразливостей, що забезпечують відтворюваність, синтаксичну коректність та різноманітність вихідного коду.

Аналіз останніх досліджень і публікацій

Стан формування датасетів для смарт-контрактів еволюціонував від невеликих наборів до масивних корпусів та синтетичних колекцій з ін'єкцією вразливостей. Ранній стандарт де-факто – екосистема SmartBugs: вона поєднує невеликий точно розмічений набір (≈143 контрактів, 208 тегованих вразливостей) для точнісного бенчмаркінгу з великим корпусом SmartBugs-Wild (≈47 398 контрактів із Ethereum), призначеним для масштабних експериментів. Такий дуальний дизайн дав спільноті як чисті еталони, так і реалістичний «шумний» контекст, однак залишив відкритим питання про баланс класів (рідкісні категорії залишаються недостатньо представленими) [3].

Новіший підхід – опора на реальні аудиторські звіти. DAppSCAN (TSE'24) системно зібрав і промаркував слабкості, пов'язавши їх з вихідним кодом та байткодом, спираючись на 1199 звітів 29 команд. Переваги підходу – валідація «з поля» та трасування до першоджерел (звіти), що суттєво підвищує довіру до міток; втім, охоплення класів і різноманітність прикладів все ще залежать від того, що реально потрапляє в аудит [4].

Паралельно з'явилися дуже великі, але переважно безрозміткові корпуси для pre-training та бенчмаркінгу інструментів – зокрема DISL (514 506 унікальних Solidity-файлів, усі верифіковані на mainnet). Такі набори критично важливі для репрезентативності та відтворюваності ML-досліджень, але самі по собі не знімають проблему дисбалансу/браку ground truth і часто потребують подальшого лейблінгу чи слабко-контрольованих евристик [5].

Щоб здобути масштаб із мітками, використовують авто-розмітку аналізаторами з majority voting. Прикладом є ScrawlD, де вразливості позначаються консенсусом кількох інструментів [6]. Це дає тисячі реальних прикладів з помірною ціною розмітки, проте вносить «label noise» (похибки/упередження інструментів), який доводиться компенсувати перевірками підмножин або поєднувати з «чистішими» джерелами міток.

Другий напрям – синтетичні датасети через ін'єкцію. SolidiFI показав, що систематичні мутації коду дозволяють будувати контрольовані колекції для чесної оцінки детекторів; при цьому автори акцентують на необхідності перевіряти компільованість і потенційну експлуатованість ін'єкцій (не всі штучні дефекти є реально небезпечними) [7]. Свіжі роботи розвивають цю лінію (наприклад, MuSe, 2025) й оптимізують оператори ін'єкцій під поширені категорії слабкостей, але центральним залишається компроміс між реалістичністю модифікацій і масштабом [8].

Сумарно, сучасний консенсус виглядає так: потрібні гібридні підходи, що поєднують реалістичність великих корпусів і керованість синтетичних ін'єкцій для балансування рідкісних класів; аудиторські джерела (як-от DAppSCAN) бажано інтегрувати як «якісний еталон»; опис наборів має відповідати сучасним стандартам прозорості (datasheets/data statements), які фіксують мотивацію, походження, процедури розмітки/ін'єкції, обмеження та відомі упередження – це підвищує відтворюваність і придатність датасетів для подальших досліджень [9, 10].

Формулювання мети дослідження

Мета дослідження полягає у підвищенні об'єктивності та достовірності навчання і тестування методів виявлення уразливостей у смарт-контрактах. Для її досягнення запропоновано створення збалансованого та відтворюваного датасету шляхом автоматизованої ін'єкції уразливостей за допомогою двох підходів – детермінованого та LLM-керованого. Такий датасет забезпечує рівномірне представлення різних класів уразливостей і створює контрольовані умови для коректного порівняння ефективності існуючих і нових інструментів аналізу безпеки [11].

Викладення основного матеріалу дослідження

Результати збору вихідних кодів смарт-контрактів

У ході дослідження було зібрано та проаналізовано вихідні коди смарт-контрактів на Solidity із двох основних джерел:

Etherscan API – актуальні верифіковані контракти з Ethereum mainnet;

SmartBugs Wild – відкритий датасет реальних контрактів із маркованими уразливостями.

Після попередньої обробки та аналізу інструментом Oyente було сформовано корпус, розподіл якого за категоріями уразливостей наведено в Таблиці 1.

Ці дані демонструють значну перевагу вразливостей типу integer overflow та integer underflow над іншими, а також порівняно малу кількість контрактів з вразливістю re-entrancy.

Таблиця 1

Розподіл контрактів за категоріями вразливостей

Категорія	Кількість контрактів
Integer Underflow	23 702
Integer Overflow	31 061
Timestamp Dependency	1 460
Re-entrancy Vulnerability	310
Без уразливостей (None)	12 829

Детермінований алгоритм ін'єкції вразливостей у смарт-контракти

Запропонований метод забезпечує детермінований підхід до автоматизованого впровадження уразливостей у вихідний код смарт-контрактів без використання машинного навчання. Алгоритм базується на статичному аналізі коду та пошуку шаблонів (pattern matching) для визначення місць, придатних для модифікації. Це дозволяє передбачувано та повторювано модифікувати код: при однакових вхідних даних результати ін'єкції завжди будуть тотожними, оскільки випадковість у процесі відсутня. Алгоритм не виконує сам контракт, а лише аналізує його статичну структуру і застосовує стандартизовані зміни, що гарантує збереження синтаксичної коректності коду після модифікації.

Метод базується на трьох ключових принципах:

– детермінованість: алгоритм завжди видає однаковий результат для одних і тих самих вхідних даних. Поведінка повністю передбачувана, оскільки у процесі модифікації відсутні випадкові фактори;

– статичний аналіз: виявлення потрібних місць для ін'єкції відбувається шляхом аналізу структури коду без виконання смарт-контракту. Алгоритм парсить вихідний код Solidity, щоб знайти специфічні конструкції та патерни, і перевіряє синтаксичну коректність зроблених змін;

– орієнтація на шаблони: для кожного типу уразливості визначено характерні шаблони коду. Алгоритм систематично шукає в контракті такі шаблонні конструкції і вносить стандартизовані зміни. Це забезпечує уніфікованість ін'єкцій та полегшує валідацію отриманих результатів.

Виявлення кандидатів (загальний випадок)

Алгоритм парсить контракт і відбирає функції-кандидати, у яких одночасно наявні: релевантні структури стану (напр., балансний mapping), операції з коштами (функції на кшталт withdraw/transfer, або зовнішні виклики переказу Ether), перевірки перед дією (require/assert), ефекти оновлення стану (напр., зменшення балансу). Лише функції, що відповідають усім критеріям цільового класу уразливості, передаються на етап модифікації.

Стратегії модифікації наявної функції (на прикладі reentrancy)

Якщо у функції вже є перевірка балансу, відправлення коштів і подальше оновлення стану, виконується цілеспрямована перестановка/заміна операцій:

– безпечний переказ (*transfer/send*) замінюється на низькорівневий виклик *call*{*value*:...}(«») із перевіркою success;

– порядок «Checks-Effects-Interactions» свідомо порушується: зовнішня взаємодія виконується до оновлення стану, що відкриває вікно для повторного входу.

```
1  // Було (безпечніше)
2  require(balances[msg.sender] >= amount);
3  payable(msg.sender).transfer(amount);
4  balances[msg.sender] -= amount;
5
6  // Стало (вразливо)
7  require(balances[msg.sender] >= amount);
8  (bool success, ) = msg.sender.call{value: amount}("");
9  require(success, "Transfer failed");
10 balances[msg.sender] -= amount;
```

Рис. 1. Мінімальний фрагмент зміни (ілюстративно) в коді

Мінімальний фрагмент зміни (ілюстративно) зображено на рисунку 1.

Стратегії додавання шаблонної функції (на прикладі reentrancy)

Якщо у вихідному контракті немає відповідної функції (наприклад, відсутній механізм виведення коштів), додається мінімальна вразлива функція з коректними сигнатурами та допоміжною логікою поповнення балансу. Приклад додавання шаблонної функції в код наведено на рисунку 2.

```
function withdraw(uint256 amount) public {
    require(balances[msg.sender] >= amount, "Insufficient balance");
    (bool success, ) = msg.sender.call{value: amount}(""); // зовнішній виклик до
    оновлення стану
    require(success, "Transfer failed");
    balances[msg.sender] -= amount; // оновлення після взаємодії
}

function deposit() public payable {
    balances[msg.sender] += msg.value;
}
```

Рис. 2. Приклад додавання шаблонної функції в код

Додавання виконується так, щоб не порушувати існуючу функціональність і компіляцію.

Валідація

Після модифікації проводиться: синтаксична перевірка (успішна компіляція solc цільової версії), підтвердження властивості уразливості статичним аналізом (наявність зовнішнього виклику до ефектів, очікувані патерни), перевірка сумісності (збереження базової логіки та сигнатур).

Математична модель детермінованого підходу

Вхідні дані: C – вихідний смарт-контракт, V – тип уразливості (наприклад, reentrancy), $P_V = \{p_1, \ldots, p_n\}$ – набір шаблонів для виявлення коду, придатного до інжектування вразливості.

1. Пошук кандидатів

Знаходимо множину функцій, що відповідають усім патернам:

$$F_V(C) = \{f \in C \mid \ \forall p \in P_V : match(p, f) = 1\}.$$

2. Вибір і модифікація

Вибираємо найпридатнішу функцію f^* за фіксованими правилами та застосовуємо оператор модифікації:

$$M\left(f^*, V\right) \to f_v',$$

де f_v' – версія з уразливістю.

3. Інжекція в контракт

$$I(C,V) = \begin{cases} C' = \left(C \setminus f^*\right) \cup f_v', якщо\ F_V\left(C\right) \neq \varnothing, \\ C' = C \cup F_{new}\left(V\right), інакше. \end{cases}$$

Якщо підхожа функція є – модифікуємо її, інакше додаємо нову шаблонну уразливу функцію.

Метод ін'єкції вразливостей у смарт-контракти на основі великих мовних моделей

Запропонований метод реалізує автоматизоване впровадження вразливостей у вихідний код смарт-контрактів із використанням великих мовних моделей (Large Language Models, LLM). На відміну від детермінованих патерн-орієнтованих алгоритмів, цей підхід забезпечує контекстно-залежні зміни, які зберігають первинну функціональність контракту і виглядають природно інтегрованими у вихідний код. Метою є формування різноманітних і реалістичних датасетів для досліджень у сфері виявлення вразливостей у смарт-контрактах.

Метод складається з чотирьох основних компонентів:

1. модуль попередньої обробки контрактів – оптимізує код для відповідності обмеженням моделі;
2. інтелектуальний інжектор вразливостей – використовує LLM для внесення змін у код;
3. модуль парсингу та валідації результатів – перевіряє правильність і якість модифікацій;
4. конвеєр генерації датасету – зберігає результати у стандартизованому форматі.

Попередня обробка контракту

Щоб задовольнити обмеження LLM (максимальна довжина запиту, обсяг токенів), код проходить оптимізацію: видалення коментарів, видалення порожніх рядків, усікання надмірно великих контрактів до граничного розміру з коректним закриттям структур. Функція оптимізації забезпечує синтаксичну завершеність навіть після обрізання коду.

Формування запиту (prompt engineering)

Система будує спеціалізований запит до LLM, що містить:

– рольову інструкцію: модель позиціонується як «дослідник безпеки»;

– чіткі обмеження: не створювати нових контрактів, модифікувати безпосередньо в наявному коді, зберігати логіку;

– вимоги до формату виходу: структура з повним модифікованим контрактом і виділеним фрагментом змін;

– опис цільової вразливості.

Генерація модифікованого контракту

Для генерації використовується модель GPT-4.1-mini з параметрами:

– temperature = 0.8 (баланс варіативності та стабільності),
– max_tokens = 4000,
– оптимізовані витрати та продуктивність.

Модель інтегрує вразливість згідно з вказівками, намагаючись мінімально змінювати код і зберігати його функціональність.

Валідація та парсинг відповіді

Після отримання відповіді виконується:

– вилучення Solidity-блоків із використанням регулярних виразів;
– нормалізація коду (усунення різниці у форматуванні);
– перевірка наявності вразливого фрагмента у повному коді;
– відхилення результатів, що не відповідають вимогам.

Конвеєр обробки

Основний цикл роботи системи включає:

1. Завантаження та перевірку контрактів.
2. Пропуск файлів > 10 КБ.
3. Виключення вже оброблених контрактів (SHA1-хешування).
4. Генерацію вразливої версії через LLM.
5. Парсинг і валідацію.
6. Збереження у форматі JSON з полями: contract – повний код, vulnerability – тип вразливості, vulnerable_snippet – змінений фрагмент.

Математична модель

Нехай: C – вихідний контракт, V – тип вразливості, AI – мовна модель.

Функція оптимізації:

$$O(C) = \begin{cases} C', якщо |C| \le MAX\_SIZE \\ truncate(C, MAX_{SIZE}), інакше. \end{cases}$$

Генерація запиту:

$$P(C, V) = f_{prompt}(O(C), config(V)).$$

Відповідь моделі:

$$R = AI(P(C, V)).$$

Функція валідації:

$$\mathrm{Val}(\mathrm{R}) = \begin{cases} \{contract, snippet\}, якщо\ snippet \in contract, \\ \varnothing, інакше. \end{cases}$$

Оцінка успішності ін'єкції:

$$S = \frac{\sum_{i=1}^{n} 1\left[Val\left(R_i\right) \neq \varnothing\right]}{n},$$

де n – кількість оброблених контрактів.

Створення збалансованого датасету та його переваги

В результаті застосування розроблених методів ін'єкції було сформовано штучний збалансований датасет смарт-контрактів, що містить по 1000 прикладів для кожної з п'яти категорій (таблиця 2).

Таблиця 2

Структура створеного датасету

Категорія	Кількість контрактів
Integer Overflow	1000
Integer Underflow	1000
Timestamp Dependency	1000
Re-entrancy	1000
Безпечні контракти	1000
Разом	5000

Переваги створеного датасету:

– баланс класів – рівна кількість прикладів для кожної категорії дозволяє уникнути класового дисбалансу, що характерний для реальних даних;

– контрольованість вразливостей – для кожного контракту точно відомо, який тип уразливості він містить (або відсутній), що спрощує підготовку навчальних та тестових вибірок;

– реалістичність коду – модифікації виконані так, щоб зберегти логіку та структуру реальних смарт-контрактів;

– відтворюваність – процес ін'єкції повністю автоматизований і може бути повторений для створення додаткових або розширених версій датасету;

– придатність для навчання моделей – забезпечує оптимальні умови для тренування і тестування алгоритмів виявлення уразливостей у смарт-контрактах.

Таким чином, отриманий збалансований датасет є важливим внеском у дослідження безпеки смарт-контрактів, оскільки поєднує переваги реальних вихідних кодів та контрольованих умов ін'єкції уразливостей.

Висновки

У роботі представлено два взаємодоповнювальні підходи до ін'єкції вразливостей у смарт-контракти Solidity для формування збалансованого датасету. Детермінований алгоритм, заснований на статичному аналізі та шаблонних перетвореннях, забезпечує відтворюваність і синтаксичну коректність, тоді як LLM-керований інжектор (GPT-4.1-mini, temperature = 0.8) створює контекстно узгоджені та реалістичні модифікації, підвищуючи різноманітність прикладів. Обидва методи інтегровані в єдиний конвеєр з нормалізацією, дедуплікацією та багаторівневою валідацією, що дозволило побудувати збалансований датасет п'яти класів, у якому всі типи вразливостей представлені рівномірно.

Практична цінність полягає в можливості використовувати набір як навчально-тестову основу для інструментів статичного аналізу та ML/LLM-моделей, а також як міст між синтетичними та реальними даними. Обмеження роботи стосуються стохастичності LLM і потреби додаткового підтвердження експлуатованості частини ін'єкцій у динамічних сценаріях.

Список використаної літератури

1. Tereshchenko O. I., Komleva N. O. Vulnerability Detection of Smart Contracts Based on Bidirectional GRU and Attention Mechanism // Communications in Computer and Information Science. 2023. Vol. 1980. Springer, Cham. DOI: https://doi.org/10.1007/978-3-031-48325-7_21

2. Tereshchenko O. I., Komleva N. O. Identification and Localization of Vulnerabilities in Smart Contracts Using Attention Vectors Analysis in a BERT-Based Model // Radio Electronics, Computer Science, Control. 2024. № 3. С. 173–184. DOI: https://doi.org/10.15588/1607-3274-2024-3-15

3. Ferreira J. F., Cruz P., Durieux T., Abreu R. SmartBugs: A Framework to Analyze Solidity Smart Contracts // ASE 2020. DOI: https://doi.org/10.1145/3324884.3415298

4. Zheng Z., Su J., Chen J., Lo D., Zhong Z., Ye M. DAppSCAN: Building Large-Scale Datasets for Smart Contract Weaknesses in DApp Projects // IEEE Transactions on Software Engineering. 2024. URL: https://doi.org/10.1109/TSE.2024.3383422

5. Morello G., Eshghie S., et al. DISL: Fueling Research with a Large Dataset of Solidity Smart Contracts : [препринт] // arXiv : [cs.SE]. 2024. URL: https://doi.org/10.48550/arXiv.2403.16861

6. Yashavant C. S., Kumar S., Karkare A. ScrawlD: A Dataset of Real-World Ethereum Smart Contracts Labelled with Vulnerabilities : [препринт] // arXiv : [cs.SE]. 2022. URL: https://doi.org/10.48550/arXiv.2202.11409

7. Ghaleb A., Pattabiraman K. SolidiFI: An Automated and Systematic Approach for Evaluating Smart Contract Static Analysis Tools // Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA). 2020. DOI: https://doi.org/10.1145/3395363.3397376

8. Iuliano G., et al. Automated Vulnerability Injection in Solidity Smart Contracts (MuSe) : [препринт] // arXiv : [cs.CR]. 2025. URL: https://doi.org/10.48550/arXiv.2504.15948

9. Gebru T., Morgenstern J., Vecchione B., Vaughan J. W., Wallach H., Daumé III H., Crawford K. Datasheets for Datasets // Communications of the ACM. 2021. Vol. 64, № 12. P. 86–92. DOI: https://doi.org/10.1145/3458723

10. Bender E. M., Friedman B. Data Statements for Natural Language Processing: Toward Mitigating System Bias and Enabling Better Science // Transactions of the Association for Computational Linguistics (TACL). 2018. Vol. 6. P. 587–604. DOI: https://doi.org/10.1162/tacl_a_00041

11. Chang S., Zhang Y., Yu M., Jaakkola T. S. Invariant Rationalization : [препринт] // arXiv : [cs.LG]. 2020. URL: https://doi.org/10.48550/arXiv.2003.09772

References

1. Tereshchenko, O. I., & Komleva, N. O. (2023). Vulnerability Detection of Smart Contracts Based on Bidirectional GRU and Attention Mechanism. Communications in Computer and Information Science, 1980. Springer, Cham. https://doi.org/10.1007/978-3-031-48325-7_21

2. Tereshchenko, O. I., & Komleva, N. O. (2024). Identification and Localization of Vulnerabilities in Smart Contracts Using Attention Vectors Analysis in a BERT-Based Model. Radio Electronics, Computer Science, Control, (3), 173–184. https://doi.org/10.15588/1607-3274-2024-3-15

3. Ferreira, J. F., Cruz, P., Durieux, T., & Abreu, R. (2020). SmartBugs: A Framework to Analyze Solidity Smart Contracts. In Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE 2020). https://doi.org/10.1145/3324884.3415298

4. Zheng, Z., Su, J., Chen, J., Lo, D., Zhong, Z., & Ye, M. (2024). DAppSCAN: Building Large-Scale Datasets for Smart Contract Weaknesses in DApp Projects. IEEE Transactions on Software Engineering. https://doi.org/10.1109/TSE.2024.3383422

5. Morello, G., Eshghie, S., et al. (2024). DISL: Fueling Research with a Large Dataset of Solidity Smart Contracts [Preprint]. arXiv:2403.16861. Retrieved from: https://doi.org/10.48550/arXiv.2403.16861

6. Yashavant, C. S., Kumar, S., & Karkare, A. (2022). ScrawlD: A Dataset of Real-World Ethereum Smart Contracts Labelled with Vulnerabilities [Preprint]. arXiv:2202.11409. Retrieved from: https://doi.org/10.48550/arXiv.2202.11409

7. Ghaleb, A., & Pattabiraman, K. (2020). SolidiFI: An Automated and Systematic Approach for Evaluating Smart Contract Static Analysis Tools. In Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2020). https://doi.org/10.1145/3395363.3397376

8. Iuliano, G., et al. (2025). Automated Vulnerability Injection in Solidity Smart Contracts (MuSe) [Preprint]. arXiv:2504.15948. Retrieved from: https://doi.org/10.48550/arXiv.2504.15948

9. Gebru, T., Morgenstern, J., Vecchione, B., Vaughan, J. W., Wallach, H., Daumé, H. III, & Crawford, K. (2021). Datasheets for Datasets. Communications of the ACM, 64(12), 86–92. https://doi.org/10.1145/3458723

10. Bender, E. M., & Friedman, B. (2018). Data Statements for Natural Language Processing: Toward Mitigating System Bias and Enabling Better Science. Transactions of the Association for Computational Linguistics (TACL), 6, 587–604. https://doi.org/10.1162/tacl_a_00041

11. Chang, S., Zhang, Y., Yu, M., & Jaakkola, T. S. (2020). Invariant Rationalization [Preprint]. arXiv:2003.09772. Retrieved from: https://doi.org/10.48550/arXiv.2003.09772

Дата першого надходження рукопису до видання: 25.09.2025
Дата прийнятого до друку рукопису після рецензування: 21.10.2025
Дата публікації: 28.11.2025