# Machine Learning for Vulnerability Detection in Smart Contracts

## A Comparison of Approaches

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Software Engineering and Internet Computing**

eingereicht von

**Mag. (FH) Stephan Klein**
Matrikelnummer 12114759

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dr. Gernot Salzer
Mitwirkung: Ass.Prof.in Dr.in Monika di Angelo

Wien, 30. Juni 2025

| Stephan Klein | Gernot Salzer |
|---|---|

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

# Machine Learning for Vulnerability Detection in Smart Contracts

## A Comparison of Approaches

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

**Diplom-Ingenieur**

in

**Software Engineering and Internet Computing**

by

**Mag. (FH) Stephan Klein**
Registration Number 12114759

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dr. Gernot Salzer
Assistance: Ass.Prof.in Dr.in Monika di Angelo

Vienna, June 30, 2025

Stephan Klein    Gernot Salzer

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

# Erklärung zur Verfassung der Arbeit

Mag. (FH) Stephan Klein

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 30. Juni 2025

Stephan Klein



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

# Acknowledgements

First and foremost, I would like to thank my advisors, Gernot and Monika, for their guidance and patience in navigating a long search for the right topic—ultimately enabling me to pursue a subject I truly care about. Their steady encouragement and objective feedback, coupled with providing compute for my tests, were invaluable.

Also, I want to thank the authors of VulHunter and MANDO, the two publications I finally evaluated in depth as part of my research. Many authors do not openly publish the technical implementation of their research—but those two research teams did 'go the extra mile' to enable other researchers like me to scrutinize their work. Not only did they publish the source code, but also hands-on documentation and tools to enable others to independently run the models and validate smart contracts from end to end.

# Kurzfassung

Smart Contracts (Blockchain-Programme) sichern mittlerweile beträchtliche Vermögenswerte und sind daher ein attraktives Ziel für Angreifer. Dabei stellt Reentrancy eine wichtige Klasse an Schwachstellen dar. Gleichzeitig führt das wachsende Interesse an maschinellem Lernen (ML) zu einer Vielzahl von Vorschlägen für automatisierte Schwachstellenerkennung als Alternative zu klassischen Analysemethoden. Bestehende Übersichtsarbeiten dazu führen verschiedene Methoden an, Klassifikationen bleiben jedoch oft ungenau oder unvollständig, und vergleichende empirische Evaluierungen sind rar. Um diese Lücken zu schließen, gehen wir als Teil dieser Arbeit folgendermaßen vor:

(1) Zur Bewertung des **Stands der Technik** führen wir eine systematische Literaturrecherche durch und leiten daraus eine Taxonomie ab, die Lernparadigmen mit konkreten Modellfamilien verknüpft, welche in der Smart-Contract-Analyse zum Einsatz kommen. (2) In einer **vergleichenden Modellanalyse** untersuchen wir zwei repräsentative ML–basierte Detektoren—MANDO-HGT (basierend auf einem Graph Neural Network) und VulHunter (basierend auf einem Bi-LSTM)—durch Auswertung der jeweiligen Publikationen und zugehörigen Open-Source-Repositories. (3) In einer **empirischen Testreihe** integrieren wir die ausgewählten Tools in die Testumgebung *SmartBugs* und führen drei Experimente durch: (I) Test der Modelle mit einer Auswahl von minimalen Reentrancy-Beispiel-Contracts (II) Tests mit einem größeren Real-World Datensatz; und (III) Retraining der Modelle am gleichen Datenset.

**Ergebnisse.** Das originale VulHunter ist konkurrenzfähig, bleibt jedoch beim Recall hinter klassischen Analysetools wie *Slither* und *Mythril* zurück. Das MANDO-Basismodell zeigt eine hohe Übererkennung (hoher Recall, geringe Präzision) und MANDO-HGT bleibt hinter den originalen Ergebnissen seiner Autoren zurück. Nach dem Retraining beider Modelle mit identischen Daten übertrifft VulHunter auf einem Holdout-Set von 120 Contracts klassische Analysetools bei F1 und Präzision, jedoch immer noch nicht beim Recall. Insgesamt legen unsere Ergebnisse nahe, dass ML ein sinnvoller ergänzender Ansatz zur traditionellen Schwachstellenanalyse sein kann—insbesondere bei sorgfältigem Instanzentwurf sowie verlässlichen Labels.

**Beiträge.** (i) eine klar strukturierte Taxonomie von ML-Techniken zur Erkennung von Smart-Contract-Schwachstellen; (ii) ein detaillierter Modellvergleich von verschiedenen ML-basierten Ansätzen (iii) eine Erweiterung des *SmartBugs*-Frameworks um ausgewählte ML-Tools; sowie (iv) eine empirische Evaluation ML-basierter Werkzeuge.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

# Abstract

Smart contracts (blockchain programs) now safeguard substantial assets and, as recent exploits show, remain attractive targets for attackers. Reentrancy persists as a damaging class of vulnerabilities. In parallel, enthusiasm for machine learning (ML) has driven a surge of proposals for automated vulnerability detection as an alternative to conventional methods. Yet the evidence base is uneven: surveys catalog methods, yet produce inaccurate or incomplete taxonomies. Comparable tool-level insights and empirical evaluations are scarce, and labeling practices remain opaque.

This thesis addresses that gap in three steps: (1) For assessing the **state-of-the-art**, we conduct a systematic literature review and derive a taxonomy that connects learning paradigms to concrete model families applied to smart-contract analysis. (2) In a comparative **paper-and-code analysis**, we study two representative ML-based detectors—MANDO-HGT (based on a graph neural network) and VulHunter (based on a Bi-LSTM)—by extracting data from the respective publications and inspecting their open-source repositories, making explicit how datasets are comprised, how instances are constructed and how information flows through model layers. (3) In an **empirical evaluation**, we integrate the selected tools into the *SmartBugs* test environment and perform three experiments: (I) a minimal reentrancy suite (260 contracts) to isolate the signal; (II) a larger, noisier benchmark (987 contracts); and (III) a fair, same-data retraining (591 contracts) to separate data effects from architectural effects.

**Findings.** Pretrained VulHunter is competitive but trails static analysis tools like Slither and Mythril in recall; the original MANDO base model over-flags (high recall, low precision); and MANDO-HGT underperforms its reported results on our datasets. After retraining both tools on identical data, VulHunter achieves the best overall balance on a 120-contract holdout, surpassing conventional analyzers in F1/precision—though not in peak recall. Overall, our results suggest that ML is a viable complement to traditional analyzers for reentrancy—especially when instance design is careful and labels are reliable.

**Contributions.** (i) a clarified, usage-oriented taxonomy of ML techniques for smart-contract vulnerability detection; (ii) an implementation-grounded, tool-level comparison that makes instance construction and model behavior explicit; (iii) an extension of the SmartBugs framework with a set of ML tools and updates of conventional tools; and (iv) an empirical evaluation of ML-based tools.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

# Contents

**Kurzfassung** v

**Abstract** vi

**Contents** vii

**1 Introduction** 1
1.1 Problem Statement and Motivation . . . . . 1
1.2 Aim of the Thesis and Expected Results . . . . . 2
1.3 Methodology . . . . . 3
1.4 Related Work . . . . . 3
1.5 Context within the Master Program . . . . . 4

**2 Systematic Literature Review** 5
2.1 Planning Phase . . . . . 5
2.2 Keyword Selection and Collection of Results . . . . . 6
2.3 Study Selection . . . . . 6

**3 State of the Art** 9
3.1 Taxonomy of ML techniques . . . . . 9
3.2 Extracting ML paradigms and used models from relevant research . . 10
3.3 Discussion of Results . . . . . 12

**4 Analysis and Comparison of Selected Tools** 18
4.1 Scope . . . . . 19
4.2 Data and Ground Truth . . . . . 20
4.3 Input and Output of Inference . . . . . 21
4.4 Construction and Representation of Instances . . . . . 23
4.5 Model Architecture . . . . . 25

**5 Evaluation of Selected Tools** 28
5.1 Original Evaluation from the Authors of the Models . . . . . 28
5.2 Experiment Design . . . . . 30
5.3 Experiment I: Evaluation on a Minimal Dataset . . . . . 33



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

5.4 Experiment II: Evaluation on a Larger Dataset with Known Ground Truth 35
5.5 Experiment III: Retraining the Models on the Same Dataset . . . . . . 40

**6 Conclusion 44**

**Appendix 47**
SLR Queries . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
Inference Reports . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48

**Overview of Generative AI Tools Used 50**

**List of Figures 51**

**List of Tables 52**

**Bibliography 53**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 1

# Introduction

## 1.1 Problem Statement and Motivation

Blockchain programs—also known as smart contracts—in particular those for Ethereum-based blockchains, have grown significantly in popularity in recent years. Decentralized Finance (DeFi) applications, which manage the assets of users with smart contracts, are especially security-sensitive. At the top of the last cryptocurrency rally in 2022, DeFi smart contracts managed 141 Bn USD considering the top 10 EVM based blockchains [def24].

The held assets have also been a target of hackers. In total, over 8 Bn USD have been stolen by hackers from blockchain applications in the largest 100 documented hacks of the last years (over all chains) [rek24]. The World Economic Forum identifies security measures as one of the biggest hurdle for the widespread adoption of smart contracts [WEF24].

Research on preventing, identifying, and correcting EVM smart contract vulnerabilities offers a promising opportunity to improve the security of blockchain applications in general. Research accompanied by an open source release of a tool demonstrating the technique in a proof-of-concept is especially valuable. Such tools can assist developers to detect and fix vulnerabilities, preventing hacks and the loss of users' funds as a consequence.

Rameder, di Angelo and Salzer reviewed automatic vulnerability analysis literature and tools up to Jan 2021 and classified the used methods into three main groups [RdAS22, p. 10ff]: *(1) Static Code Analysis (e.g. Symbolic Execution and Abstract Interpretation), (2) Dynamic Code Analysis (e.g. Fuzzing), (3) Formal Specification and Verification*

Other methods are collected in a 'Miscellany' group, and include machine learning techniques like long short-term memory (LSTM) modeling, convolutional neural networks or



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

N-gram language models. Out of 140 tools included in the review only 10 employed a machine learning technique.

Since this review, the AI Hype Cycle, a term coined by Gartner[Gar23], reached its 'peak of inflated expectations' in 2023 when generative AI applications like LLMs and image generators based on diffusion models were becoming mature and increasingly popular. This development was preceded by a sharp increase in AI-related scientific publications. While from 2010 to 2016 the number of AI publication in computer science was relatively stable at about 100.000 per year, the number more than doubled until 2023 to over 240.000 [MFP+25, p. 29]. Our literature review indicates that this trend also applies to smart contract vulnerability research.

The high expectations regarding AI are often not matched by reliable and measurable results. Appropriately, the next phase in Gartner's hype cycle is called 'Trough of Disillusionment'. By identifying and testing promising tools, as well as evaluating and comparing the different techniques they employ, we aim to determine how useful AI and machine learning in particular are for detecting vulnerabilities in smart contracts.

## 1.2 Aim of the Thesis and Expected Results

The work aims to achieve two major goals:

1. Summarize the status quo of research on machine learning-based smart contract analysis, focusing on the reentrancy vulnerability as a popular and relevant sample weakness.
2. Identify a concise set of promising open source detection tools accompanying publications in the field, compare their approaches and models, and finally evaluate their performance on a common dataset.

We deduct the following research question from goal 1:

- *RQ1: What machine learning (ML) techniques are being applied for the detection of vulnerabilities in smart contracts and how can they be classified ?*

We deduct the following research questions from goal 2:

- *RQ2.1: What promising research in the context of RQ1 accompanied by the release of functional open source tools for detecting the reentrancy vulnerability exist?*
- *RQ2.2: How do different approaches and ML models of the chosen publications compare?*
- *RQ2.3: How well can selected ML tools detect the reentrancy vulnerability?*



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

## 1.3 Methodology

The research questions above serve as a starting point of a Systematic Literature Review (SLR). The next step of the SLR is to deduce a set of search queries from the RQs to identify relevant publications. We aim to execute the query on a set of relevant publication databases (*ACM*, *IEEE*, *ScienceDirect* and *Google Scholar*) to extract suitable publications.

A preliminary execution shows that there exist over 250 candidate publications, therefore we will extract the state-of-the-art (RQ1) from other literature reviews and surveys on the topic. Suitable candidates are summarized in the following section.

The list of candidates will be further reduced by defining and applying selection criteria thus filtering out non-suitable publications. For identifying suitable tools (RQ2.1) we scan the reduced list of candidates for links to GitHub or Zenodo and evaluate documentation and code by visual inspection of the repositories. Finally, we choose a small set of candidate tools and publications based on:

1. the related journal or conference rating [Cor24] [Sci24]
2. the quality, documentation and suitability of the provided tool and its code repository

After publications and tools have been selected, we compare their general approach, employed techniques and model architecture to identify similarities and differences based on the literature (RQ2.2).

To address RQ2.3, we evaluate original versions of selected ML tools on two independent sets of smart contracts—a minimal set of contracts illustrating different types of reentrancy [Fü25] and a subset of *Consolidated Ground Truth (CGT)* [dAS23]. Finally, we retrain the underlying models of selected tools using a combination of these datasets to obtain a direct comparison.

## 1.4 Related Work

The results of our initial literature search include surveys and reviews related to our research questions. We group them in two categories based on the types of literature they study: (1) *General publications not focused on AI in particular*, (2) *Publications applying AI and specific ML techniques.*

The first category contains Rameder et al. [RdAS22] as described in Section 1. Also Zaazaa and El Bakkali [ZB23] conduct a systematic literature review of smart contract vulnerabilities, tools, and datasets across 107 studies from 2016 to 2021 and identify 70 frameworks of which 21 are machine learning based.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

A similar work by Chu et al. [CZD+23] identifies 20 frameworks including 6 based on deep learning (time range 2015–2022) and also inspects work on automatic vulnerability repair.

Wei et al. [WSZ+24] provides an overview on smart contract vulnerabilities, attacks, defenses, and detection methods. They also benchmark common contract analyzers versus LLMs (GPT-4o and Llama-3.1-8b) and find that GPT-4o achieved the best performance in most configurations. Contract analyzers employing ML techniques are not considered in the benchmark.

In the second category we find Jiang et al. [JCX+23] presenting a comprehensive survey on enhancing smart-contract security using machine-learning techniques. They shortlist 32 publications of proposed machine learning frameworks and classify their approaches into supervised, semi-supervised and reinforcement learning. The vast majority of tools are based on supervised learning techniques, which they briefly describe.

Kiani et al. [KS24] conduct a similar review including 55 publications. They propose a more specific classification into classical, deep learning and ensemble learning models as well as individual subcategories. Additionally, they take the issues of class imbalance and unknown vulnerabilities into consideration and extract the datasets used by the chosen publications.

Liu et al. [LWS+24] broadly survey the usage of the transformer model in blockchain technology, but dedicate a chapter to vulnerability detection. They identify seven relevant publications and categorize them into two groups: graph-based and sequence-based representation of the smart contract within the model.

None of the publications in the second category test or benchmark the identified frameworks.

## 1.5 Context within the Master Program

The proposed work builds upon various theoretical and practical knowledge acquired through the *Software Engineering and Internet Computing* curriculum. The topic can be contextualized on the intersection of blockchain and AI. Therefore, all smart contract related courses in the field of advanced security and ML related subjects in the field of algorithmics are relevant.

Additionally, the researched ML models are often combined with formal verification methods covered in the courses about formal methods and advanced software engineering. Graph theoretical concepts are fundamental as well, in case the smart contract representation in the ML model is graph-based.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

# CHAPTER 2

# Systematic Literature Review

Systematic literature reviews (SLR) aim to present a fair evaluation of a research topic by using a trustworthy, rigorous, and auditable methodology to minimize bias [Schnd]. Its key elements are outlined in Table 2.1. For us this structured approach is particularly

| Phase | Steps |
| --- | --- |
| Planning | 1. Formulate your research question |
| | 2. Establish a pilot |
| Conducting the Review | 3. Choose appropriate search keywords |
| | 4. Conduct the search and collect studies |
| | 5. Select relevant studies |
| | 6. Analyze primary studies |
| Reporting | 7. Report on the results |

Table 2.1: Phases and key steps of SLR [Schnd]

useful for comprehensively assessing the current state of ML techniques employed for vulnerability detection (RQ1), as well as for identifying the most promising ML-based tools and prototypes produced out of this body of research (RQ2).

## 2.1 Planning Phase

The planning phase of the SLR is an iterative process to revise an initial set of research questions until the understanding of the problem increases. This process also guides us to focus our work from an initial, larger set of potential research question to a final set of refined ones, as outlined in Section 1.2. Quantitatively, this refinement reduces the number of search results in academic databases from 1,296 for the initial queries to 334 for the final queries.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

## 2.2 Keyword Selection and Collection of Results

In the next phase, we deduct search queries from the research questions. For this purpose we define a query expression $Q$ as shown in Equation 2.1: a conjunction of disjunctions, where each disjunction represents one mandatory aspect $A_1 ... A_4$ of our research by choosing a set of synonyms and alternative terms.

$$\begin{aligned}
A_1 &= (\text{"learning"} \lor \text{"artificial intelligence"} \lor \text{"artificial-intelligence"} \lor \text{"ai"} \lor \text{"neural"} \lor \text{"embedding"} \lor \text{"supervised"} \lor \text{"unsupervised"}),\\
A_2 &= (\text{"smart contract"} \lor \text{"smart contracts"} \lor \text{"smart-contract"} \lor \text{"smart-contracts"} \lor \text{"ethereum"}),\\
A_3 &= (\text{"reentrancy"} \lor \text{"re-entrancy"} \lor \text{"reentrance"} \lor \text{"re-entrance"}),\\
A_4 &= (\text{"github"} \lor \text{"zenodo"}),\\
Q &= A_1 \land A_2 \land A_3 \land A_4.
\end{aligned} \tag{2.1}$$

We observe in the pilot already that a full text search for $Q$ leads to many irrelevant results. Therefore, we limit the search for aspects $A_1 ... A_3$ to the abstract and title of the publication where supported by the search engine. $A_4$ (The code repository we require), however, may occur anywhere in the document.

$Q$, representing our target query, must be adopted in practice for execution across various academic databases due to limitations such as the maximum number of supported search terms or the availability of grouping. The exact queries used for each database are listed in the Appendix.

Table 2.2: Overview of searches performed

| ID | DB | Search In ($A_1 ... A_3$) | Date | #Results |
|---|---|---|---|---|
| Q1 | ACM | Abstract | 04.11.2024 | 11 |
| Q2 | ACM | Title | 04.11.2024 | 5 |
| Q3 | IEEE | Abstract | 04.11.2024 | 47 |
| Q4 | IEEE | Title | 04.11.2024 | 22 |
| Q5 | ScienceDirect | Everywhere | 05.11.2024 | 41 |
| Q6 | Google Scholar | Title | 06.11.2024 | 208 |

The final searches, as shown in Table 2.2, were executed between 04.11 and 06.11.2024 and delivered a set of publications for further sub-selection.

## 2.3 Study Selection

After removal of duplicate results by title and applying a filter to exclude outdated work before 2018, 263 candidate publications remain for further evaluation. Already in the pilot phase, we establish that the majority of initial search results do not match our research questions. As an illustrative example, the article “Blockchain and NFTs for Trusted Ownership, Trading, and Access of AI Models” [BMY+22] fulfills our search query but is not related to our specific field of research. There are many different reasons why a search result does not match our particular area of interest. We define such reasons for exclusion or inclusion in a set of *Selection Criteria (SC)* in Table 2.3. This allows us



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Table 2.3: Selection Criteria

| ID | Inclusion Criteria | Exclusions | Inclusions |
|---|---|---|---|
| SC1 | Study proposes or advances ML-based techniques for analyzing smart contracts | 118 | 145 |
| SC2 | The techniques address the reentrancy vulnerability | 32 | 113 |
| SC3 | The smart contract itself is analyzed, not only its transactions or other metadata | 5 | 108 |
| SC4 | Full publication is available (via open or institutional access) in English | 4 | 104 |
| SC5 | A code repository was released alongside the publication on GitHub or Zenodo | 70 | 34 |
| SC6 | The repository is accessible and contains code as well as documentation | 6 | 28 |
| SC7 | The journal or conference has a high rating (CORE2023 A or higher / Scimago SJR 1500 or higher) | 16 | 12 |

to narrow down the list of publications most relevant to our goals.

While the initial queries and filtering were applied with the search engine's capabilities in a semi-automated way, the SCs are evaluated manually. To reduce the processing time, we apply SCs sequentially, i.e. $SC5$ is only tested for publications passing $SC1 ... SC4$ instead of testing all SCs on all publications.

$SC1 ... SC3$ can be tested in most cases by evaluating the abstract. $SC4$ is trivially decided negatively if the publication is not available to us in English.

For $SC5$, we first search the full publication text for a GitHub or Zenodo reference. If none is found, we use Google, combining the publication's title with "*site:github.com*" to ensure a potential post-publication code release is not missed.

We evaluate $SC6$ by visual inspection of the repository and assess the availability of code and a minimum set of instructions on how to use the tool.

After applying $SC6$, 28 candidates are left for detailed evaluation in the scope of RQ2. The final thought of the selection phase is to focus our work on the most promising research available. For this purpose we apply one last qualitative measure and determine the *Core2023* conference rating [Cor24] for conference papers or *Scimago* rating [Sci24] respectively for journal publications and choose the highest rated ones. The final set of primary studies is listed in Table 2.4.

We utilize the selected primary studies in the following ways. In Chapter 3, we describe and classify the machine learning techniques employed. In Chapter 4, we focus on a subset of tools for an in-depth inspection of the models. Finally, in Chapter 5, we independently



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

| Title | Year | GitHub Link | GitHub Stars | Core2023 Rating | Scimago SJR |
|---|---|---|---|---|---|
| Combining Graph Neural Networks With Expert Knowledge for Smart Contract Vulnerability Detection [LQW+23] | 2021 | 🔗 | 102 | | 2867 |
| Smart contract vulnerability detection using graph neural networks [ZLQ+21] | 2021 | 🔗 | 143 | A* | |
| Smart contract vulnerability detection: from pure neural network to interpretable graph feature and expert pattern fusion [LQW+21] | 2021 | 🔗 | 78 | A* | |
| Eth2Vec: Learning Contract-Wide Code Representations for Vulnerability Detection on Ethereum Smart Contracts [AYCO21] | 2021 | 🔗 | 37 | A | |
| Checking Smart Contracts With Structural Code Embedding [GJX+21] | 2021 | 🔗 | 104 | | 1868 |
| Peculiar: Smart Contract Vulnerability Detection Based on Crucial Data Flow Graph and Pre-training Techniques [WZW+21] | 2022 | 🔗 | 3 | A | |
| Cross-Modality Mutual Learning for Enhancing Smart Contract Vulnerability Detection on Bytecode [QLYH23] | 2023 | 🔗 | 26 | A* | |
| MANDO-HGT: Heterogeneous Graph Transformers for Smart Contract Vulnerability Detection [NNX+23] | 2023 | 🔗 | 18 | A | |
| VulHunter: Hunting Vulnerable Smart Contracts at EVM Bytecode-Level via Multiple Instance Learning [LLZ+23] | 2023 | 🔗 | 9 | | 1868 |
| xFuzz: Machine Learning Guided Cross-Contract Fuzzing [XYZ+24] | 2024 | 🔗 | 3 | | 2222 |
| Smart Contract Code Repair Recommendation based on Reinforcement Learning and Multi-metric Optimization [GCC+24] | 2024 | 🔗 | 11 | | 1853 |
| sGuard+: Machine Learning Guided Rule-Based Automated Vulnerability Repair on Smart Contracts [GYY+24] | 2024 | 🔗 | 6 | | 1853 |

Table 2.4: Set of primary studies and their tools

evaluate their performance.

As a by-product of our selection process, we gathered reviews and surveys, which were also part of our initial search results (See 1.4 for a summary). In the following chapter, we incorporate selected surveys to present the state-of-the-art and a classification of methods.



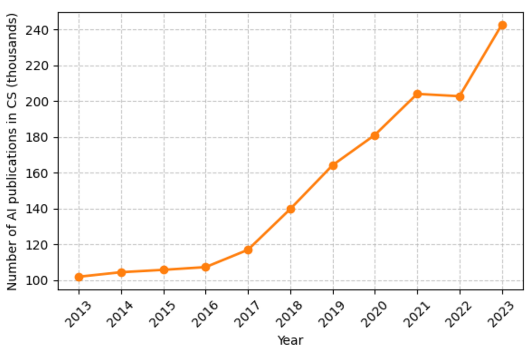
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

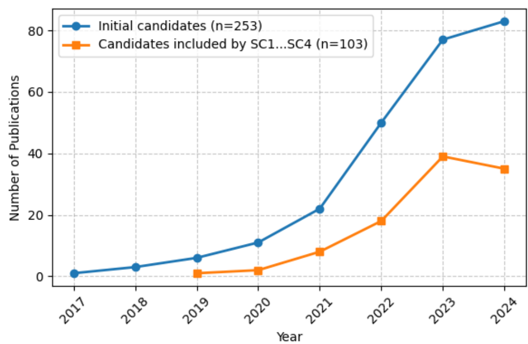
CHAPTER 3

# State of the Art

Computer science research in general, as well as research related to our particular field, shows a clear trend to employ artificial intelligence, as depicted in Figure 3.1.[1]




(a) Number of AI publications in CS worldwide, 2013-2023 [MFP+25, p. 29]




(b) Publications per year identified in our SLR

Figure 3.1: AI trend in CS

Within the broad topic of AI in computer science, we focus on machine learning in our work, because it is the most prevalent field within AI publications of recent years [MFP+25, p. 38].

## 3.1 Taxonomy of ML techniques

To answer *RQ1: What machine learning (ML) techniques are being applied for the detection of vulnerabilities in smart contracts and how can they be classified?* we need to

[1] *Note on the 2024 plateau/decline:* Our literature review was conducted between November 4 and November 6, 2024. Studies published after this period were not included in our analysis.

Bibliothek Your knowledge hub

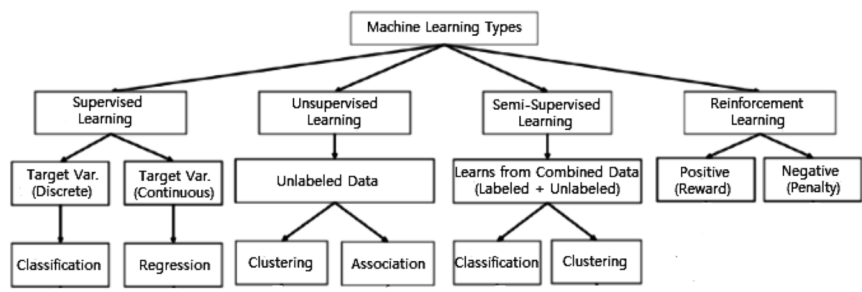



Figure 3.2: Classification of Machine Learning Techniques [Sar21b]

establish a taxonomy of ML techniques first.

ML techniques are often differentiated into the following paradigms: *supervised*, *unsupervised*, *semi-supervised*, and *reinforcement learning* [Sar21b]. This differentiation is based on the task and available knowledge, as shown in Figure 3.2. Our hypothesis is that supervised or semi-supervised ML techniques must be prevalent in research because vulnerability detection is clearly a binary classification task.

During the literature review, we find that this traditional classification is outdated, as it does not cover some emerging deep learning models. One such model we encounter is *Bidirectional Encoder Representations from Transformers (BERT)*, where a foundation model is initially pre-trained in a self-supervised way and later fine-tuned for a supervised task (i.e., smart contract vulnerability detection). Emmert-Streib and Dehmer already address this gap and suggest seven additional paradigms — one being *transfer learning (TL)* — precisely what we need to classify a model based on BERT [ED22].

Another perspective of classifying ML techniques is applied by Kiani et al. in their review about smart contract vulnerability detection based on ML [KS24]. They extract the concrete models from 55 relevant publications and group them into *classical*, *deep learning*, and *ensemble models*.

We combine the extended list of paradigms and the model classification of Kiani, as shown in Table 3.1 with the goal of classifying relevant research within this taxonomy.

## 3.2 Extracting ML paradigms and used models from relevant research

The next step to answer *RQ*1 is to retrieve the employed ML paradigms and models from relevant publications. For this purpose, we combine results from other literature reviews already mentioned in Section 1.4 with additional work identified in our own literature review.

The result is a classification of eighty relevant publications by learning paradigm (Table 3.2) and ML model (Table 3.3). Fifty-five publications were covered in the reviews by



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

| | |
|---|---|
| **ML Paradigms** | Supervised — Unsupervised — Semi-Supervised — Reinforcement Learning— Transfer Learning |
| **ML Models & Families** | |
| *Classical* | Random Forests (RF) — Support Vector Machines (SVM) — k-Nearest Neighbours (KNN) — Decision Trees (DT) — Logistic Regression (LR) — Naïve Bayes (NB) — Stochastic Gradient Descent classifiers (SGD) |
| *Deep Learning* | Graph Neural Networks (GNN) — Convolutional Neural Networks (CNN) — Long Short-Term Memory networks (LSTM) — Bidirectional Encoder Representations from Transformers (BERT) — Gated Recurrent Unit networks (GRU) — Bidirectional LSTM (Bi-LSTM) — Bidirectional Gated Recurrent Unit networks (Bi-GRU) — Deep Neural Network (DNN) — Edge-Enhanced Convolution (EEC) — Graph Convolutional Networks (GCN) — Extreme Gradient Boosting Trees (XGBT) — Heterogeneous Graph Transformer Networks (HGTN) |
| *Other NNs* | Artificial Neural Network (ANN) — Word2Vec — FastText |
| *Ensemble Learning* | Extreme Gradient Boosting (XGBoost) — Adaptive Boosting (AdaBoost), Categorical Boosting (CatBoost) — Gradient Boosting (GBoost) |

Table 3.1: Taxonomy of machine-learning paradigms and models

Kiani et al. [KS24, p. 15] and thirty-two by Jiang et al. [JCX+23, p. 21] already. Three publications not covered by them were supplemented from our SLR.[2]

Upon collecting the results from Jiang and Kiani, we apply some corrections:

1. Jiang et al. employ the traditional classification of ML paradigms and classify two BERT-based architectures as semi-supervised, which we find inaccurate, as explained in Section 3.3.5. We reclassify them under transfer-learning.

2. Jiang et al. collect the employed ML models, but in some cases, the provided information is too vague (i.e., the method is described as machine learning or deep learning only without more details). In such cases, we supplement the particular model or techniques used from the primary reference directly.

3. Kiani's classification of models contains artificial neural networks (ANN) and deep neural networks (DNN). We apply those in case a model cannot be subclassified further into a specific subclass like GNNs, CNNs, LSTMs, etc. Models that could be identified as a subclass were not counted as ANN or DNN again explicitly, even if a GNN represents both an ANN and a DNN.

[2]Many single publications were included in multiple reviews (Kiani, Jiang or our own SLR). We identified these duplicates and consider them in all quantifications provided, e.g., in Tables 3.2 and 3.3.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4. Kiani et al. differentiate supervised from deep learning models categorizing only five publications as “supervised” explicitly. We do not find this differentiation accurate because deep learning is used in many implementations for supervised learning, especially for a classification task like vulnerability detection. As a result, we reclassify relevant publications identified by them in our taxonomy.

5. Kiani et al. imprecisely classify *Word2Vec*, *FastText*, and *ANN* under deep learning. We reclassify them under *Other Neural Networks* as ANNs can be shallow or deep, and Word2Vec and FastText represent shallow networks.

6. We omit Jiang’s reference 114, as it is a literature review. No particular method is presented within this publication.

| **Model** | **n[1]** | **Reference of Kiani[2]** | **Reference of Jiang[3]** | **Primary Studies of our SLR** |
|---|---|---|---|---|
| Supervised | 73 | 9, 16, 21, 24, 25, 26, 40, 41, 42, 43, 44, 45, 48, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 65, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93 | 38, 106, 107, 108, 109, 110, 111, 112, 113, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133 | [LQW+23, ZLQ+21, LQW+21, AYCO21, WZW+21, QLYH23, NNX+23, LLZ+23, XYZ+24, GYY+24] |
| Transfer | 3 | 9, 86 | 130 | [WZW+21, QLYH23] |
| Reinforcement | 2 | | 133 | [GCC+24] |
| Semi-Supervised | 1 | | 134 | |
| Unsupervised | 1 | | 134 | [GJX+21] |

[1] number of distinct publications [2] Source: [KS24, p. 15] [3] Source: [JCX+23, p. 21]

Table 3.2: ML paradigms employed in literature for smart contract vulnerability detection

After taking over the results from Kiani and Jiang and applying the corrections as discussed, we extract paradigms and models from three additional publications identified in our review to complete the classification.

## 3.3 Discussion of Results

As expected, and as shown in the results of our classification by ML paradigm in Table 3.2, primarily supervised learning techniques are employed in seventy-three out of eighty publications for smart contract vulnerability detection. We summarize exceptions using non-supervised techniques in Section 3.3.5.

We display the employed models and model families shown in Table 3.3 also on a time scale in Figure 3.3 to identify baselines and trends.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Figure 3.3: Occurrence of ML models in selected publications over time

### 3.3.1 Classical Models

Classical methods—mostly random forests (RF), support vector machines (SVM), decision trees (DT) and k-nearest neighbor (KNN)—are prevalent in the research we review. In many instances, however, they are only supplemented, in addition to a main model (often based on deep learning), as a baseline method to evaluate the performance of the main model against. One such instance is the work by Li et al. [LLZ+23].

### 3.3.2 Deep Learning Models

In the majority of included studies, researchers decide to build elaborate deep learning models for classifying vulnerabilities in smart contracts. A majority of the models employed can be further subclassified, as shown in our taxonomy in Table 3.1. We detect the following popular subclasses in more than one publication.

**Graph Neural Networks (GNN)** are popular within our selection of research, employed in twenty publications. They subsume deep learning-based methods that operate on



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

the graph domain [ZCH+21]. GNNs use a message-passing mechanism to aggregate information from neighboring nodes, allowing them to capture the complex relationships [KPKT24]. Such an approach of direct graph representation within the model needs to be differentiated from others, where information of a graph is extracted into a regular 2D feature-matrix and processed with a classical ML model or conventional ANN[3].

Variants of GNNs include graph convolutional networks (GCN), graph attention network (GAT), and graph recurrent network (GRN), which have demonstrated ground-breaking performances on many deep learning tasks [ZCH+21]. In Chapter 4, we take a closer look at a GAT-based architecture for smart contract vulnerability detection.

**Convolutional Neural Networks (CNN)**, used in 18 publications, were originally created for image recognition by LeCun et al. [LBBH98]. During inference, a convolutional layer extracts features from an image (such as edges, endpoints, and corners) while moving a kernel (subset of $n \times n$ pixel) over the image. To process smart contracts with a CNN, for example, Hwang et al. convert the smart contract's bytecode into a fixed-size RGB image, by mapping a sequence of opcodes to the three channels (R, G, B) first. Such images can be processed by a CNN, but applying 2D kernels with a (high) stride, as in a traditional CNN, destroys semantics and context of the smart contract. They propose a different, non-stride architecture based on 1D filters instead, outperforming earlier CNN-based architectures [HCSC22, pp. 32600–32604].

**Long Short-Term Memory (LSTM)** networks are a type of recurrent neural network (RNN), introduced by Hochreiter et al. [HS97]. LSTMs are designed to avoid the exploding/vanishing gradient problem of CNNs and RNNs, which occurs in model training during backpropagation. For this purpose, regular nodes of a recurring layer are replaced with LSTM units that have three internal gates managing the flow of information and storing two different states for short- and long-term memory instead of one.

Seventeen publications included in our research employ LSTMs, and nine use the bidirectional variant. In a regular LSTM, the sequence of instances is processed in one direction, such that the model only accesses the past context when predicting the next step, whereas in Bi-LSTMs, the contexts of both directions are combined, providing a fuller understanding of the sequence [Gee24]. We describe the utilization of a Bi-LSTM network for smart contract vulnerability detection in Chapter 4.

**Gated Recurrent Unit (GRU)** networks are another popular variant of an RNN introduced by Cho et al. [CvMG+14]. They follow the same conceptual idea as an LSTM, but rely on a more streamlined unit structure. GRUs have been shown to perform better on certain smaller and less frequent datasets; however both variants of RNNs have proven their effectiveness [Sar21a, pp. 7–9].

[3] An example of such an approach is described in Chapter 3.3.3



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

### 3.3.3 Shallow NNs—Word2Vec

Word2Vec, a method in natural language processing (NLP) introduced by Mikolov et al. [MCCD13], learns compact vectors representing the meaning of words by training a shallow neural network on a corpus of text to predict which words appear near each other. These vectors capture semantic and syntactic relationships and work better than older, sparse representations in many NLP tasks like search, classification, and clustering [JM25, Chapter 6.8].

Gao et al. utilize Word2Vec and FastText, a similar technique, to detect bugs in smart contracts. First, the contracts' abstract syntax tree (AST) is normalized and serialized into sequences at a granularity of contracts, functions, or statements. Next, the sequences are used to train a Word2Vec/Fasttext model. Embeddings of vulnerable statements, functions and contracts are stored separately. Those embeddings are finally compared to embeddings of an unseen contract by calculating the Euclidean distance. The resulting similarity metric is used to indicate vulnerabilities like reentrancy [GJX+21].

### 3.3.4 Ensemble Learning Models

In general, ensemble learning is used to combine multiple *weak learners* to obtain a better classifier. We classified Random Forests (RF) under classical methods; however, it can be viewed as ensemble technique as well. By fitting several decision trees in parallel and using majority voting or averages to calculate the result, a better accuracy can typically be achieved compared to a single DT-based model.

Adaptive boosting (AdaBoost) and extreme gradient boosting (XGBoost) are the two most prevalent ensemble techniques in the publications we study. Unlike RF, which uses parallel ensembling, AdaBoost uses sequential ensembling instead. It works best on binary classification, but is sensitive to noisy data and outliers, and it can lead to overfitting.

XGBoost, like neural networks, uses gradient descent to minimize a loss function. However, while neural networks apply it to optimize weights, XGBoost applies it to sequentially build optimal decision trees. It also applies advanced regularization, which reduces overfitting [Sar21b, p. 7].

Gao et al. compare an implementation of XGBoost for trees with other tree-based classifiers to detect vulnerabilities in smart contracts. They engineer a feature set by combining Opcode Word2Vec embeddings with static code analysis features at the function level, and use it to classify five types of vulnerabilities. Their experiments show that XGBoost outperforms AdaBoost, RF, and standard decision trees overall [GYY+24].

### 3.3.5 Non-Supervised Techniques

Non-supervised techniques can be combined with supervised learning in various ways. Wu et al., for example, modify *GraphCodeBERT*, a GNN model based on BERT in their model named *Peculiar* [WZW+21]. In three self-supervised pretraining tasks, the



TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

representation from source code, data flow, and the alignment between those two is learned. The pretrained model can be reused for diverse downstream tasks such as natural language code search, clone detection, code translation, and code refinement [GRL+21]. In the case of Peculiar, ground truth labels are supplied to fine-tune the pre-trained model for vulnerability detection. This general approach is also known as transfer learning.

Semi-supervised learning (SSL), which addresses the issue of a lack of labeled data, represents another approach to combine different ML techniques. The paradigm is applied by Sun et al. for smart contract vulnerability detection. With a small dataset labeled by experts, a model is trained initially. Next, unlabeled data which can be predicted with high-confidence by the trained model, is pseudo-labeled. In a subsequent training step the pseudo-labeled instances are used to improve the model [STZ+23].

Another approach based on reinforcement learning (RL) is presented by Guo et al., where a vulnerable area is first identified in the source code and then attempted to be repaired by an agent trained with RL. The agent tries one of 15 generic edit actions distilled from a Solidity fix guide (e.g. replace *call.value*() with *send*) and a reinforcement function evaluates multiple metrics like the result of compilation and whether the vulnerability was fixed [GCC+24].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

| Model | n[1] | Ref. of Kiani[2] | Ref. of Jiang[3] | Primary Studies of our SLR |
|---|---|---|---|---|
| **Classical** | 17 | | | |
| RF | 14 | 41, 16, 21, 24, 25, 26, 43, 44, 87, 89 | 115, 113, 122, 124 | [AYCO21], [LLZ+23], [GYY+24] |
| SVM | 9 | 24, 25, 26, 43, 44, 89 | 106 | [LLZ+23], [XYZ+24] |
| DT | 9 | 16, 26, 40, 44, 87 | 122, 124 | [LLZ+23], [XYZ+24], [GYY+24] |
| KNN | 7 | 42, 24, 26, 40, 44 | 124 | [XYZ+24] |
| LR | 4 | 40, 44 | 122 | [XYZ+24] |
| NB | 2 | 44 | | [XYZ+24] |
| SGD | 1 | 52 | 126 | |
| **Deep Learning** | 61 | | | |
| GNN | 20 | 63, 93, 9, 86, 65, 61, 62, 60, 64, 68, 67, 49, 70, 66, 51, 71, 88 | 132, 129, 123, 121 | [LQW+23], [ZLQ+21], [LQW+21], [WZW+21], [QLYH23], [NNX+23], [LLZ+23] |
| CNN | 18 | 53, 54, 52, 21, 45, 49, 85, 47, 55, 56, 57, 58, 59 | 112, 118, 128, 126, 131, 107, 124 | [LLZ+23] |
| LSTM | 17 | 83, 48, 73, 45, 47, 25, 50, 69, 76, 89 | 112, 111, 116, 122, 124 | [LLZ+23], [GCC+24] |
| Bi-LSTM | 9 | 75, 90, 74 | 108, 112, 111, 131, 125, 124 | [LLZ+23], [XYZ+24] |
| BERT | 7 | 9, 45, 50, 80 | 134, 131, 130 | [WZW+21] |
| Bi-GRU | 5 | 46, 57, 84 | 112 | [LLZ+23] |
| GRU | 4 | | 112, 116, 117 | [LLZ+23] |
| DNN | 3 | 82 | 110, 119, 135 | |
| FastText | 1 | | 105 | [GJX+21] |
| EEC | 1 | | | [XYZ+24] |
| GCN | 1 | | 131 | |
| HGTN | 1 | | 120 | |
| **Other NNs** | 11 | | | |
| ANN | 9 | 79, 81, 61, 21, 82, 25, 26, 44 | 116, 119 | |
| Word2Vec | 6 | 16 | 105, 131, 109, 38 | [GJX+21], [XYZ+24], [GYY+24] |
| **Ensemble** | 11 | | | |
| XGBoost | 9 | 16, 24, 25, 87, 89 | 109, 122 | [LLZ+23], [XYZ+24], [GYY+24] |
| AdaBoost | 5 | 16, 24, 89 | 122 | [XYZ+24], [GYY+24] |
| CatBoost | 1 | | 113 | |
| GBoost | 1 | 87 | | |

[1] number of distinct publications [2] Source: [KS24, p. 15] [3] Source: [JCX+23, p. 21]

Table 3.3: ML Models used for smart contract vulnerability detection



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CHAPTER 4

# Analysis and Comparison of Selected Tools

So far, our systematic literature review has resulted in a shortlist of twelve high-quality publications, selected based on specific criteria aiming to best align with our research focus. Additionally, we integrated these with findings from existing reviews to provide a comprehensive overview of the ML paradigms and techniques employed across the field.

These results provide a broad answer to *RQ2.1: What promising research in the context of RQ1 accompanied by the release of functional open source tools for detecting the reentrancy vulnerability exist?* and *RQ2.2: How do different approaches and ML models of the chosen publications compare?* already. However, we still lack a detailed understanding of specific implementations. Aspects such as datasets and their labeling, data representation, and performance of different methods compared to conventional methods have not yet been covered.

To address this gap, we select two publications of our shortlist for a closer inspection. The selection is based on the availability of the following artifacts in the respective project's repositories:

1. The complete source code for model training and inference
2. The datasets used for training the model
3. A functional program for testing smart contracts—in Solidity or bytecode representation—using the model trained by the authors

If these supplementary artifacts and data are provided by the original authors, we are not only able to evaluate the model's performance independently, but can also address gaps in the respective publications when comparing the proposed approaches.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

We provide an overview about the two publications selected for a detailed comparison in Table 4.1.

| **Tool Name** | **MANDO** | **VulHunter** |
|---|---|---|
| References | Nguyen et al. [NNX+22] (Base Model) and [NNX+23] (HGT) | Li et al. [LLZ+23] |
| Source Repositories | Main Project 🔗<br>Inference App 🔗 | Main Project 🔗 |
| Model Type / Model Subtype | Graph Neural Network (GNN) / *Base Model:* Graph Attention Network (GAT)<br>*Enhanced Model:* Heterogeneous Graph Transformers (HGT) | Recurrent Neural Network (RNN) / Bag-instance & self-model attentions based Bi-directional Long Short-Term Memory ($Bi^2$-LSTM) |

Table 4.1: Publications and their tools selected for detailed analysis and evaluation

MANDO was originally published by Nguyen et al. in 2022 [NNX+22]. The same group subsequently developed an enhanced version, MANDO-HGT, published in 2023 [NNX+23]. We consider the enhanced version for the comparison in this chapter.

## 4.1 Scope

Both tools were developed to detect vulnerabilities in smart contracts. MANDO detects seven vulnerability types, derived from the labels in its training datasets. In contrast, VulHunter implements detection for thirty-one types. Although the authors do not state this explicitly when introducing their detectors, the majority of these types (23 out of 31) correspond to bugs defined by well-known contract analyzers such as Slither and Mythril. The authors later note that Slither was used—alongside other tools—to supplement ground truth labels for the contracts of the dataset [LLZ+23, p. 4896].

The only two vulnerabilities they are both trained on are *Reentrancy (SWC-107)* and *Unchecked Low Level Calls (SWC-104)*. A full list including a mapping to SWC[1] and DASP[2] classification is provided in Table 4.2.

Regarding the granularity of detection, MANDO supports both 'Coarse-Grained' (contract-level) and 'Fine Grained' (node-level) detection. Additionally, it is able to map nodes back to lines in the Solidity source, after they have been flagged by the detector.

VulHunter directly infers only fine-grained labels for potential contract execution sequences. It 'extracts the key opcodes with large weights in sequences' from the result, and 'locates the defective contract source code statements by mapping from the assembly

[1] https://swcregistry.io/
[2] https://dasp.co/



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

language [to the} source code file' [LLZ+23, p. 4888]. The instance with the largest probability is taken over as the detection result of the contract [LLZ+23, p. 4894].

| MANDO notation | VulHunter notation | SWC | DASP |
|---|---|---|---|
| Arithmetic | | SWC-101 | DASP-3 |
| Reentrancy | Reentrancy-eth | SWC-107 | DASP-1 |
| Access Control | | — | DASP-2 |
| Denial of Service | | SWC-113, SWC-128 | DASP-5 |
| Front Running | | SWC-114 | DASP-7 |
| Time Manipulation | | SWC-116 | DASP-8 |
| Unchecked Low-Level Calls | Unchecked-lowlevel | SWC-104 | DASP-4 |
| | Controlled-array-length | — | — |
| | Suicidal | SWC-106 | DASP-2 |
| | Controlled-delegatecall | SWC-112 | DASP-2 |
| | Arbitrary-send | SWC-105 | DASP-2 |
| | TOD | SWC-114 | DASP-7 |
| | Uninitialized-state | SWC-109 | — |
| | Parity-multisig-bug | — | — |
| | Incorrect-equality | SWC-132 | DASP-5 |
| | Integer-overflow | SWC-101 | DASP-3 |
| | Unchecked-send | SWC-104 | DASP-4 |
| | Tx-origin | SWC-115 | DASP-2 |
| | Locked-ether | — | DASP-5 |
| | Boolean-cst | — | — |
| | Erc721-interface | — | — |
| | Erc20-interface | — | — |
| | Costly-loop | SWC-128 | DASP-5 |
| | Timestamp | SWC-116 | DASP-8 |
| | Block-other-parameters | SWC-116 | DASP-6 |
| | Calls-loop | SWC-113 | DASP-5 |
| | Low-level-calls | — | — |
| | Erc20-indexed | — | — |
| | Erc20-throw | — | — |
| | Hardcoded | — | — |
| | Array-instead-bytes | — | — |
| | Unused-state | — | — |
| | Costly-operations-loop | — | — |
| | Send-transfer | — | — |
| | Boolean-equal | — | — |
| | External-function | — | — |

Table 4.2: Mapping of vulnerabilities supported by MANDO and VulHunter to SWC and DASP classifications

## 4.2 Data and Ground Truth

As shown in Table 4.3, MANDO-HGT's dataset for vulnerable contracts contains the 143 manually annotated contracts of *Smartbugs Curated* and 350 of the *SolidiFi-Benchmark*, where contracts were synthetically injected with bugs [GP20]. This combined dataset contains line-level labels and is used to train the model for fine-grained detection. For



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

coarse-grained detection, the combined vulnerable dataset is concatenated with a random selection of benign contracts from the *SmartBugs Wild* dataset maintaining 1:1 ratio for each vulnerability type.

VulHunter uses a significantly larger dataset originally provided by Zhuang et al. in their early work on vulnerability detection using GNNs [ZLQ+21]. While the authors of VulHunter indicate that ground truth labels are available in this dataset [LLZ+23, p. 4895], we were unable to locate them in the original source[3]. They further state: 'we have manually checked and supplemented these labels based on the verification results of multi-methods such as SmartFast, Slither and Oyente'[LLZ+23, p. 4896]; however, they do not provide further details on how those results were aggregated into final labels.

| Dataset | Cat. | Total SCs | Subset SCs* | Ratio B:V | Ratio Train:Test | Loops** |
|---|---|---|---|---|---|---|
| **Mando** | | | | | | |
| 1. SB Curated[1] | V | 143 | 71 | 1:1 | 7:3 | 20 |
| 2. SolidiFI[2] | V | 350 | | | | |
| 3. SB Wild[3] | B | 2 742 | | | | |
| **VulHunter** | | | | | | |
| 1. ESC[4] | V + B | 38 600 | 446 | 2:1, 5:1 | 4:1 | 5 |
| 2. TSE[5] | V + B | 526 | | | | |

[*] Number of vulnerable contracts used for training the reentrancy detection model.
[**] Number of executed train+test loops with random sampling of train/test split.
[1] [FCDA20] - https://github.com/smartbugs/smartbugs-curated
[2] [GP20] - https://github.com/DependableSystemsLab/SolidiFI-benchmark
[3] [FCDA20] - https://github.com/smartbugs/smartbugs-wild
[4] [ZLQ+20] - https://github.com/Messi-Q/Smart-Contract-Dataset
[5] [CXL+22] - https://github.com/Jiachi-Chen/TSE-ContractDefects
V=Vulnerable, B=Benign

Table 4.3: Datasets used for training selected ml tools

## 4.3 Input and Output of Inference

To evaluate the tool's input and output, we examined the source code available in the repositories listed in Table 4.1 and executed the respective implementation for inference. For this purpose, the authors of MANDO provide a web application in a separate repository, while VulHunter can be started via the command line.

### Input

Both tools accept Solidity files as input for vulnerability detection. MANDO's web application sends the uploaded source code file to a Python-based backend for inference. While the MANDO-HGT model is theoretically capable of processing bytecode, the provided inference tool does not support it out of the box. VulHunter, in addition to Solidity files, also accepts compiled bytecode (`.bin`) or opcode (`.evm`) files.

[3] See GitHub link in Table 4.1



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

**Output**

MANDO delivers the result of the smart contract analysis via a JSON response, and presents it in its web application. In contrast, VulHunter generates a PDF report which is internally rendered from a JSON structure as well. We provide exemplary screenshots of both results in the Appendix.

MANDO's result contains a JSON object for each tested type of vulnerability with the following fields:

| | |
|---|---|
| **type** | $\in$ `MANDO_BUGTYPES` (See table 4.2) |
| **graph runtime** | $\in \mathbb{N}$ |
| **node runtime** | $\in \mathbb{N}$ |
| **number of bug node** | $\in \mathbb{N}$ |
| **number of normal node** | $\in \mathbb{N}$ |
| **vulnerability** | $\in \{0, 1\}$ |
| **results** | $= [(id_i,\ lines_i,\ vuln_i)]_{i=1}^{n}$<br>$id_i \in \mathbb{N},\ lines_i \subseteq \mathbb{N},\ vuln_i \in \{0,1\}$ |
| **smart contract length** | $\in \mathbb{N}$ |
| **graph** | control flow graph |
| **heatmap** | data for a bug heatmap |

Besides a binary vulnerability classifier on contract level given in *vulnerability*, the *results* element provides an identification of vulnerable and non-vulnerable lines in the contract.

VulHunters's result has the same top-level structure (one array entry for each type of vulnerability) with the following fields:

| | |
|---|---|
| **title** | $\in$ `VULHUNTER_BUGTYPES` (See table 4.2) |
| **severity** | $\in$ {`Low, Medium, High, Info, Opt`} |
| **confidence** | $\in$ {`maybe, probably, certain`} |
| **instances** | $= [c_i]_{i=1}^{m},\ c_i \in \{0,1\}$ ... is instance $i$ vulnerable |
| **description** | a text of the general vulnerability description |
| **positions** | $= [(id_j,\ \Delta_j)]_{j=1}^{\lvert c_i=1 \rvert}$<br>$id_j$...id of vulnerable instance<br>$\Delta_i = [(imp_{jk},\ map_{jk},\ loc_{jk},\ pos_{jk})]_{k=1}^{3}$<br>$imp_{jk} \in [0,1]$ ... importance i.e. weight<br>$map_{jk} \in \{\texttt{true}, \texttt{false}\}$ ... if the location can be mapped to sourcecode<br>$loc_{jk} \in \{\ \texttt{auxdata} \vee (\text{begin} : (\ell, c),\ \text{end} : (\ell, c))\ \mid\ \ell, c \in \mathbb{N}\}$<br>$pos_{jk} \in \mathbb{N}$ ... index of position in the input instance<br><br>$\ell$...line<br>$c$...column |



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

The result includes an array *instances* of flags, indicating the result of classification for each extracted opseq-sequence, a process described in the next chapter. In *positions*, the three highest weights (indicating importance) are provided for each vulnerable instance, each corresponding to a particular position in the `OPCODE` input sequence. In case it can be mapped to a position in the Solidity file, the line and column are provided for the beginning and end of the vulnerable position; otherwise, "auxdata" is returned

## 4.4 Construction and Representation of Instances

Supervised ML models for classification are trained with large labeled datasets. They are often structured in the shape of a ($m$ rows) × ($n$ columns) matrix $X$—the vertical dimension representing instances or samples, and the horizontal dimension representing features—and a vector $y$ of length $m$ providing the ground truth label for each instance.

To learn the intricate logical connections within a smart contract instance, its source code or bytecode cannot be directly passed as a one-dimensional vector of features; therefore, instance building is a crucial part of any advanced ML model, especially for classifying code.

MANDO-HGT and VulHunter start this process with similar approaches by building control flow graphs (CFGs), but end up with completely different data structures for learning. This is dictated by their choice of NN model—GNNs ingest graphs, and LSTMs ingest sequential data.

### 4.4.1 VulHunter

Instance construction starts with the contract's bytecode, as provided by the Solidity compiler. Next, VulHunter employs the evm_cfg_builder library[4], developed by *Trail of Bits*, to obtain a Control Flow Graph (CFG). The graph consists of vertices representing straight-line code sequences without branches, called basic blocks, and directed edges representing branches from one exit to one entry of neighboring basic blocks [LLZ+23, p. 4891].

From the CFG, a DFS-based algorithm returns opcode sequences by following paths of the CFG, starting from all basic blocks without incoming edges. To avoid space and performance explosion, the algorithm is limited by the following parameters [LLZ+23, p. 4892].

- $n_{seq}$: the number of sequences returned for each contract—considered after DFS, choosing the $n_{seq}$ longest sequences returned by the DFS
- $n_{cycle}$: the maximum number of cycles, i.e., a limit of how often the same block is included in the sequence repeatedly

[4] https://github.com/crytic/evm_cfg_builder



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- $n_{block}$: maximum number of individual blocks traversed

The standard VulHunter model, as tested by our benchmark, was trained with $n_{seq} = 10$, $n_{cycle} = 2$, and $n_{block} = 32$, as stated by the authors. We confirm this by inspecting the source code and finding the parameters either hardcoded or set as default values of program arguments. We want to point out that such limits, in particular the block limit of 32, represent an important limitation of VulHunter, as vulnerabilities after 32 blocks in the sequence will not be visible to the model.

The $n_{seq}$ sequences of T opcodes $opseq = \{x_1, ..., x_T\}$ and the bag instance $C$, identifying the contract, are generated for each contract of the dataset and fed into the Bi$^2$-LSTM model for classification and training. For the latter, additionally, the binary label is provided to the model. The first layer of the NN encodes each *opseq* into a real-valued bytecode sequence. The LSTM is a type of RNN and therefore designated for such sequential data. The bi-directional training process can learn forward and backward relationships from the sequences.

For model training, the ground truth (binary vulnerable/benign labels) is only available per contract, not per sequence. Therefore, the contract's label is applied to all generated sequences. This inaccuracy represents a source of noise, addressed by the authors as 'problem of missing fine-grained labels' [LLZ+23, pp. 4892–4893]. The problem is mitigated at loss calculation during model training, where 'two losses are fused to constitute the Bag-instance hybrid attention mechanism, enabling the model to identify malicious instances in the bag guided by the bag/contract identification' [LLZ+23, p. 4894].

### 4.4.2 MANDO-HGT

MANDO-HGT supports two types of instances: source code- and bytecode-based heterogeneous control-flow graphs (HCFGs)—*heterogeneous*, because different node and edge types represented in the graph retain the syntax and semantics of the code.

For the first type, MANDO-HGT does in fact build a graph from Solidity functions and instructions—unlike VulHunter, which immediately compiles the source code to bytecode for further processing. At first, MANDO-HGT generates a CFG for each function of all contracts in the Solidity file separately, including vertices for each type of Solidity statement (node types) and edge types $e \in \{True, False, Next\}$. Then, the CFGs of individual functions are 'fused' by adding edges between them and to additional vertices representing invoked Solidity language functions and external contracts, derived from a call graph generated by Slither[5] [NNX+23, p. 337].

For the second type, the CFG based on bytecode is generated by using Ethersolve [CCCP21], which provides the same kind of CFG as the evm_cfg_builder library previ-

[5] `https://github.com/crytic/slither/wiki/Printer-documentation#call-graph`



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

ously discussed, but employs a different technique[6].

Two artifacts are extracted from the graph: (1) node features, which are used as initial embeddings, and (2) pairs of meta-relations, structuring the data for input into the model. A meta-relation of an edge $e = (s, t)$ from a source node $s$ to a target node $t$ is defined as

$$\langle \tau(s),\, \phi(e),\, \tau(t) \rangle,$$

with $\tau(s)$ and $\tau(t)$ representing the node type of $s$ and $t$, respectively, and $\phi(e)$ representing the edge type of $e$. The goal of processing the meta-relations in pairs, is to learn the *attention* between a target node $t$ and two of its neighbor source nodes *s1* and *s2* [NNX+23, p. 338].

The embeddings representing the nodes of the meta-path relations in the first layer can be of heterogeneous nature depending on the problem domain. The authors of MANDO-HGT experiment with three variants: (1) one-hot encoded node types; (2) heterogeneous node embeddings of *metapath2vec*; (3) heterogeneous node embeddings of *node2vec* [NNX+23, p. 338]. MANDO's inference application tested in chapter 5 supports only the first variant.

## 4.5 Model Architecture

The MANDO-HGT model closely follows the original HGT model as presented by Hu et al. [HDWS20]. The authors of MANDO-HGT make use of the property of HGT that any downstream task can be attached onto the HGT—binary classification in our case. The HGT model is a GNN version of the original transformer model first described in the publication *Attention is all you need* [VSP+17], which was built for NLP tasks like translation or next-token prediction.

The authors of Vulhunter do not provide a reference to any base architecture of their Bi$^2$-LSTM model, but their implementation reveals that for the Bi-LSTM layers Pytorch's LSTM[7] is employed, which is based on the work of Sak et al. [SSB14].

The NN model architectures as presented by the authors are shown side by side in Figure 4.1. As a guide to interpreting the figures, we provide an overview of the NN's respective layers in table 4.4. It aims to show how information is extracted from contract instances and flows between the layers within the respective model for one forward pass from top to bottom—and vice versa during backpropagation. The information provided for each layer was mostly extracted from the respective publication but supplemented from the respective implementation where necessary.

Regarding the granularity of data flow through each layer, MANDO-HGT processes the CFG of a single contract from the dataset in a single forward pass, while VulHunter

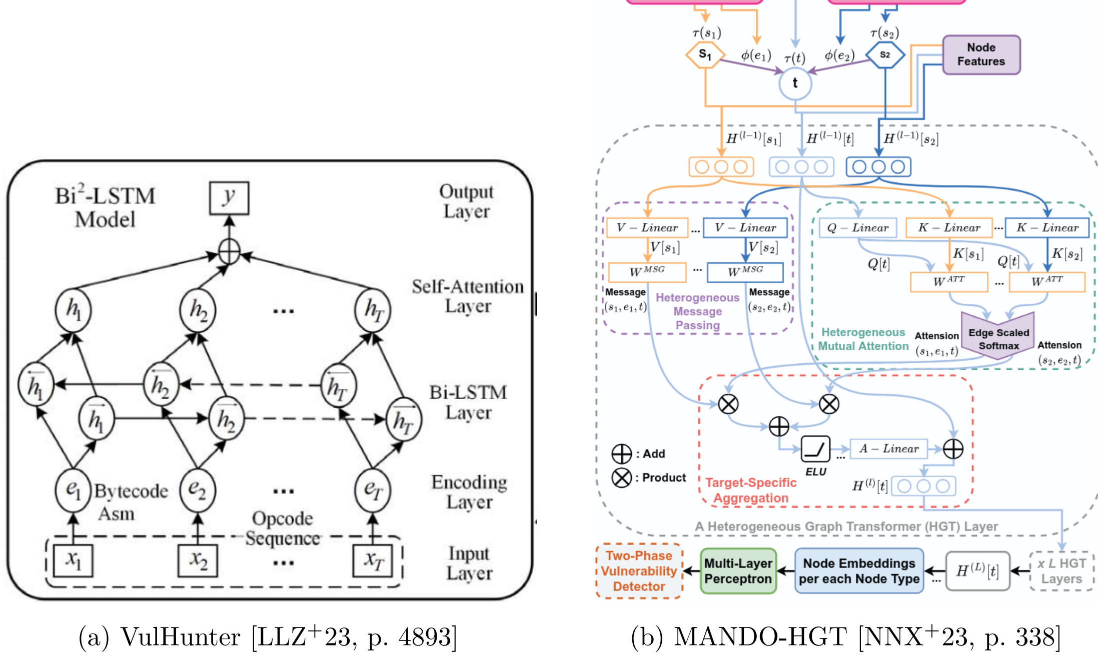
[6] Ethersolve uses symbolic execution; evm_cfg_builder uses value set analysis (VSA) to resolve jump destinations

[7] `https://docs.pytorch.org/docs/stable/generated/torch.nn.LSTM.html`



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

processes one batch of opcode-sequences. These instances and their construction are described in the previous chapter. For further implementation details of each component, we refer to the respective chapters of MANDO-HGT [NNX+23, pp. 338-339] and VulHunter [LLZ+23, pp. 4892-4894].




(a) VulHunter [LLZ+23, p. 4893]

(b) MANDO-HGT [NNX+23, p. 338]

Figure 4.1: NN model architectures



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

| Layer / Module | Purpose | Input → Output[1] | Notes |
|---|---|---|---|
| **MANDO-HGT Graph Neural Network** | | | |
| Meta-Relation Extractor (pre-processing) | See chapter 4.4.2 | CFG $G$ → Meta-relations $\mathcal{M} = \{\langle\tau(s), \phi(e), \tau(t)\rangle : e \in E(G)\}$ | – |
| Node Features Extractor (pre-processing) | | CFG $G$ → Initial node embeddings $H^0$ | Modes: (1) one-hot encoded node-types, (2) metapath2vec, or (3) node2vec |
| $L$ HGT Layers ($L$=2 per default) | Learn attention between all $s, t \in \mathcal{M}$ | $(H^0, \mathcal{M}, G)$ → Output node embeddings after $L$ stacked layers $H^L$ | Components: (1) meta relation-aware heterogeneous mutual attention, (2) heterogeneous message passing, and (3) target-specific heterogeneous message aggregation. |
| MLP Classification Head *Coarse-grained* | Graph-Level vulnerability classification | $H^L$ → Confidence score for each graph $\hat{y}_G \in [0,1]$ | MLP with softmax activation function |
| MLP Classification Head *Fine-grained* | Node-Level vulnerability classification | $H^L$ → Confidence score for each node $\{\hat{y}_t\}_{t \in V(G)} \mid \hat{y}_t \in (0,1)$ | |
| **VulHunter Bi$^2$-LSTM** | | | |
| Instance Builder / Input Layer | See chapter 4.4.1 | CFG (bytecode) → $n_{seq}$ opcode-sequences $opseq = \{x_1, ..., x_T\} \mid x_i \in OPCODES$ | For simplification we assume single opseqs are processed in one forward pass[2] |
| Encoding Layer | | $opseq$ → bytecode-sequences $C_{opseq} = \{e_1, ..., e_T\} \mid e_i \in BYTECODES$ | |
| Bi-LSTM Layer | Capture bidirectional contextual relationships within the sequences | $C_{opseq}$ → Hidden-state sequence $H \in \mathbb{R}^{d \times T}$ [3] | $T$ represents the time dimension in a LSTM. Input is processed sequentially. |
| Self-Attention Layer | Learn *importance* $\alpha$ of each element of a sequence and collapse T for next layer | $H$ → $(\alpha \in (0,1)^T, h^* \in (-1,1)^d)$ | |
| Vulnerability Classifier | *Training:* Calculate loss on opseq (instance) level *Inference:* Classify opseq | $h*$ → $(loss_{ins},\ \hat{y} = [1-p, p] \in [0,1]^2)$ | The result of fusing $loss_{ins}$ with $loss_{bag}$ is used for back-propagation. |
| Bag-Instance Hybrid Attention | *Training:* Calculate loss on contract level (1 contract = 1 bag) *Inference:* Classify bag | $(loss_{ins}^{n_{seq}}, \hat{y})$ → $(loss_{bag},\ \hat{y}_{bag} = [1-p_{bag}, p_{bag}] \in [0,1]^2)$ | |

[1] Input and output for a single forward pass. Inputs specify external information used in the layer, and outputs specify information passed to following layers. We do not specify weights, intermediary results and other learned parameters internal to the respective layer. Variable notation was taken over from the respective publications and supplemented where necessary.
[2] This is also assumed in the VulHunter publication, whereas in the implementation opseq's are processed in batches
[3] $d$ ... size of the hidden dimension; defaults to $512 \times 2$ in VulHunter implementation.

Table 4.4: Layers of compared NN models



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

CHAPTER 5

# Evaluation of Selected Tools

To address the final RQ 2.3: *How well can selected tools detect the reentrancy vulnerability?*, we summarize the evaluation results of the original authors first. Then, we design three experiments to extend the evaluation from different points of view and provide additional insights.

I. Evaluation on a minimal dataset (260 contracts)—consisting of smart contracts illustrating different reentrancy scenarios in vulnerable and benign configurations—to obtain an indication of how well the models can detect the reentrancy vulnerability in an environment with minimal noise from unrelated code

II. Evaluation on a larger, diverse dataset (987 contracts), where the ground truth was determined by earlier research to obtain performance results on longer contracts including noise

III. Direct comparison of the learning capabilities of different architectures by training the models on the same dataset

## 5.1 Original Evaluation from the Authors of the Models

### 5.1.1 Setup

The authors of MANDO-HGT create a balanced dataset by combining 423 vulnerable contracts with an equal number of randomly sampled clean contracts from SmartBugs Wild. More precisely, the authors picked those 423 contracts from the datasets SmartBugs curated and SolidiFI (cf. Table 4.3) that can be processed by their graph generation libraries. For each vulnerability type, a separate model is trained and tested. In the case of reentrancy, this process yields a dataset containing only 71 vulnerable and 71 benign contracts. The number of examples provided to the model for training is further



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

reduced by a training/test split of 7:3. The final results were obtained by averaging the outcomes of 20 executions, each using a different random seed. Training and evaluation are executed for graphs extracted from source code and bytecode separately [NNX+23, p.340].

In the case of VulHunter, the underlying dataset for training and testing is much larger. However, only a subset is used for training and testing the reentrancy model, as shown in Table 4.3. The publication does not disclose these statistics, but we can deduce from their sources on GitHub[1] that 446 vulnerable contracts are selected from datasets 1 and 2 and mixed in 2:1 and 5:1 ratios of benign to vulnerable contracts. They report the average results based on five random seeds of the train/test split [LLZ+23, p.4896].

### 5.1.2 Discussion of Results

The results of the evaluation of both tools with the described setup, model variations, and the results of the best competitor tested by the authors are shown in Table 5.1. We highlighted the best result for each metric.

The results of MANDO are hardly comparable to the results of VulHunter because of different datasets, ground-truth labeling strategies, and also because of the chosen metrics. The authors of MANDO choose to report only macro F1 and binary F1 for the vulnerable class of the base model, and only the former for the improved HGT model.[2]

MANDO attains high performance scores—in particular the HGT variant ($F1 \approx 0.95$)—which seems very promising. By contrast, the included test of conventional tools (*Securify*, *Mythril*, *Slither*, *Manticore*, *SmartCheck* and *Oyente*) achieved substantially lower scores ($F1 = 0.23$), although in our experimens some of these tools performed considerably better and MANDO considerably worse.

We note that the final training dataset for reentrancy contains only 50 contracts, which we consider insufficient for producing a robust model. Furthermore, reporting only the F1 metric offers an incomplete view of the tool's performance. Precision and recall, considered individually, are crucial for different stakeholders: recall is vital for detecting as many vulnerabilities as possible, while precision is important for minimizing false positives that may frustrate developers. A comprehensive evaluation should therefore report both metrics and, ideally, also include measures that reflect performance on the negative class, such as specificity.

The authors of VulHunter, on the other hand, include recall and precision, but only for the vulnerable class, so we cannot directly obtain the performance of negative-class prediction as well.

[1] We provide a Jupyter notebook for dataset statistics calculation in our fork of VulHunter on GitHub at `https://github.com/stephan-klein/VulHunter/blob/main/dataset_stats.ipynb`

[2] A footnote by the authors of MANDO-HGT states that macro F1 was very close to the F1 of the bug label and therefore omitted [NNX+23, p. 340].



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

| Configuration | A | P | R | F1 | Macro F1 |
|---|---|---|---|---|---|
| **Mando** | | | | | |
| MANDO Graph Level | - | - | - | 76.09 | 75.80 |
| MANDO Node Level | - | - | - | 86.40 | **80.78** |
| MANDO-HGT G.L. source code | - | - | - | **94.78** | - |
| MANDO-HGT G.L. bytecode | - | - | - | 87.97 | - |
| MANDO-HGT N.L. | - | - | - | 92.59 | - |
| Securify | - | - | - | 23.0 | - |
| **VulHunter** | | | | | |
| Vulhunter 2:1 | 95.29 | **95.06** | 90.59 | 92.77 | - |
| Vulhunter 5:1 | 96.46 | 90.36 | 88.24 | 89.29 | - |
| Slither 5:1 | **99.02** | 94.44 | **100.00** | **97.14** | - |

* Binary F1-Score for the vulnerable class
Sources: [NNX+22, Table II & IV], [NNX+23, Table II & III], [LLZ+23, Table VI & VII]
A=Accuracy, P=Precision, R=Recall, and F1 - all measured for the vulnerable class

Table 5.1: Evaluation results of original authors, including best model configurations for reentrancy and best competitor tool

An open question regarding VulHunter concerns the authors' exact ground-truth labeling strategy, as also outlined in Chapter 4.2. Publishing the precise labeling procedure would enable a more accurate assessment of the tool's real-world performance. We suspect a strong dependency of the final labels on the results of Slither, given its near-perfect score in the authors' evaluation.

## 5.2 Experiment Design

Conducting our own experiments requires running the selected tools—MANDO, MANDO-HGT, and VulHunter—as well as additional conventional tools for comparison, on the same dataset and collecting results in a systematic way. Di Angelo et al. provide the modular execution framework *SmartBugs 2.0* [dADFS23], an ideal candidate for our test environment.

### 5.2.1 Tools under Test

We extend SmartBugs by providing version updates of selected conventional tools that we want to include in our evaluation: Mythril 0.24.8, Slither 0.11.3, and Solhint 5.1.0, in our fork of SmartBugs on GitHub[3]. Also, we provide the integration of the selected ML tools in a separate branch, *mltools*, with the following variations:

1. *vulhunter*: VulHunter supporting source code and bytecode analysis.

[3] https://github.com/stephan-klein/smartbugs



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2. *vulhunter-verify*: Starts VulHunter with `-verify` flag, activating the feasibility validation. This module, based on an SMT solver, reports whether the bytecode sequence instance is feasible, or not. Infeasible instances will be corrected as benign, thus reducing false positives [LLZ+23, p.4894].

3. *mando*: The MANDO inference application with activated base model, supporting source code based detection via command line.

4. *mando-hgt*: The MANDO inference application with activated HGT model supporting source code based detection via command line.

5. *mando-orig*: The original MANDO inference application with activated base model supporting source code based detection via an application deployed on a web server (REST-Endpoint). Omitted in our experiments because of long setup and runtime for each detection. A sanity check on a smaller dataset was executed to confirm our improved version *mando* reports identical results.

### 5.2.2 Modifications of ML-based Tools

For Experiments I and II, we want to assess how well the authors' holistic approaches—including dataset selection, labeling, preprocessing, architecture and fine-tuning—are suitable for detecting the reentrancy vulnerability. Therefore, we do not train the selected ML tools on additional datasets and instead use snapshots of the models trained originally. In contrast, for Experiment III, we retrain the models on the same dataset for a direct comparison of how the models perform given the same knowledge.

Primarily for compatibility with the SmartBugs framework, we adopted and improved the tools as follows. A full list of changes can be obtained by comparing our forks, referred to in the footnotes, with their respective origin repository.

*vulhunter/vulhunter-verify* [4]*:*

1. Replaced the included solc binaries, some of which were corrupted and produced segmentation faults on execution

2. Provided a Dockerfile

3. Added missing fonts, which prevented the PDF report from being printed

4. Added logging of exceptions in case of solc compilation errors

5. Added writing of the JSON vulnerability report to a file additionally to stdout

*mando* [5]*:*

[4] `https://github.com/stephan-klein/VulHunter`
[5] `https://github.com/stephan-klein/ge-sc-machine`



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

1. Migrated the inference endpoint's code to a Python script, `inference.py`, which can be directly executed and writes a JSON file as output to eliminate the overhead of running a full web server for testing each smart contract
2. Load solc version from an environment variable so it can be set externally
3. Improved exception handling

*mando-hgt* [6]*:*

1. Switched to the HGT model by uncommenting the prepared configuration in `utils.py` and replacing the paths to the relevant model snapshots in `config.py`
2. Fixed the implementation in `model_hgt.py` so that it does not crash when encountering an unknown edge type in the input graph

### 5.2.3 Dummy Baseline Classifiers

In addition to the results of conventional and ML-based tools, our evaluation includes two dummy classifiers implemented with *scikit-learn*[7], serving as baselines for comparison: (1) *coin-flipper*, which applies the `uniform` strategy to generate random classifications without considering class balance. (2) *over-achiever*, which applies the `constant` strategy with `constant=True` to label every contract as vulnerable, thereby yielding perfect recall.

### 5.2.4 Metrics Calculation

We evaluated our binary classifier using scikit-learn's metrics `accuracy_score`, `precision_score`, `recall_score`, and `f1_score`[8]. For *precision*, *recall*, and *F1* we used scikit-learn's defaults, i.e., `average='binary'` and `pos_label=1`, so these metrics target the vulnerable (positive) class. For *specificity*, we calculate the `average='binary'` recall of `pos_label=0`. Consequently, the formulas for calculating the chosen metrics are:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}, \qquad \text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN},$$

$$\text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2TP}{2TP + FP + FN}, \qquad \text{Specificity} = \frac{TN}{TN + FP}$$

[6] `https://github.com/stephan-klein/ge-sc-machine/tree/hgt`

[7] `https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html`

[8] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html`
`https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html`
`https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html`
`https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html`



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

## 5.3 Experiment I: Evaluation on a Minimal Dataset

The goal of our first experiment is to answer the question of how much the trained models learned about the reentrancy vulnerability specifically. Therefore, we need to define *reentrancy* first. In summary, a reentrancy attack occurs when an external contract calls back into a vulnerable function before its previous execution finishes, allowing the attacker to manipulate state or drain funds in an unexpected order.

Extended definitions and examples of the attack are available from Loi et al. [LCO+16] and Rodler et al. [RLKD18], for example. Additionally, based on the latter, Fürst describes four different categories of reentrancy attacks in their thesis *Reentrancy in Ethereum Smart Contracts* [Fü25]:

1. *Same-function reentrancy:* The attacker re-enters the same function that initiates the external call.
2. *Cross-function reentrancy*: Entering a different function in the same contract, which can also lead to inconsistent states.
3. *Delegated reentrancy*: In cases where the attacking contract is executed via `DELEGATECALL` or `CALLCODE` opcodes.
4. *Create-based reentrancy*: When the execution of a newly created contract's constructor is triggered by the `CREATE` opcode, which re-enters the victim directly or via further external calls.

They also provide a dataset of 260 Ethereum smart contracts[9] (a result of perpetuating minimal example contracts of the four categories in vulnerable and non-vulnerable forms over different Solidity versions) and test them with selected tools— an improved version of Oyente[10], Mythril 0.23.1, an updated version of Sailfish[11], and eThor 2023—for detecting reentrancy. The dataset is class-imbalanced in a 9:4 ratio (benign vs. vulnerable).

We merge these test results with the results of testing the same 260 contracts in our test setup for mando, mando-hgt, vulhunter, vulhunter-verify and our selected conventional tools mythril-0.28.8, slither-0.11.3, and solhint-5.1.0.

Finally, we also integrate Fürst's measurements[12] with the following considerations:

1. Fürst tests smart contracts in source code and bytecode representations and provides separate results as well as an aggregation. As our selected tools are tested with their Solidity source code only, we choose the source code-based results of Fürst as well.

[9] `https://github.com/lmfuerst/DA_Testdaten`
[10] Available at `https://github.com/smartbugs/oyente/tree/update`
[11] Available at `https://hub.docker.com/r/lmfue/sailfish`
[12] Available at `https://github.com/lmfuerst/da-results-to-csv`



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2. As an exception, we take over bytecode-based results for eThor, as no source code-based results are available.
3. We omit the results for Mythril as we include our own results from a newer version of Mythril.
4. We calculate the F1 score from recall and precision provided by Fürst: $F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

Overall results are shown in Table 5.2. Table 5.3 includes separate results per type of reentrancy, and Figure 5.1 shows individual results for each Solidity version.

### 5.3.1 Overall Results

Mando classifies only 10 of 260 contracts as benign (the actual number is 180), which leads to near perfect recall but poor overall performance. The resulting metrics are nearly identical to those of the over-achiever dummy classifier, which flags every contract as vulnerable.

Mando-hgt, in contrast, classifies the majority of contracts as benign but makes several mistakes in both classes and produces the second-highest number of false negatives among all test tools. Consequently, its recall and F1 scores lag behind both baseline dummy classifiers. In terms of the relative number of mistakes over all contracts, as expressed by the accuracy metric, mando-hgt outperforms vulhunter and nearly performs as well as Mythril.

Vulhunter produced the same predictions regardless of whether verification of feasible instances was enabled or not.[13] It outperforms the baseline dummy classifiers for all metrics and finds more actual vulnerabilities than any tested conventional tool apart from Slither, while—unlike mando—also performing decently in the negative class. However, as it also produces more false positives than conventional tools excluding eThor, its precision and specificity trail behind.

### 5.3.2 Results per Solidity Version

Due to continuously changing language features and the introduction of new opcodes, which might affect analysis results, the 260 contracts of the dataset consist of 52 base contracts duplicated for the 5 Solidity versions, namely 0.4.26, 0.5.17, 0.6.12, 0.7.6, and 0.8. Between versions, the corresponding Solidity code differs only in cases where certain functions were deprecated or adapted. For example, version 0.5 uses `msg.sender.call.value(500000)("")`, and version 0.6 uses the equivalent `msg.sender.call{value: 500000}("")` instead [Fü25, pp. 40-41].

When testing contracts of Solidity version 0.4, vulhunter outperforms most tested conventional tools in terms of recall, F1 and accuracy. Mando-hgt also performs best on version

[13] Therefore, we omit results for vulhunter-verify in tables and diagrams for Experiment I



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

| Tool | P | R | F1 | A | S | TP | FP | TN | FN | E/T |
|---|---|---|---|---|---|---|---|---|---|---|
| mando | 0.32 | **1.00** | 0.48 | 0.35 | 0.06 | 80 | 170 | 10 | 0 | 0/260 |
| mando-hgt | 0.39 | 0.21 | 0.27 | 0.65 | 0.85 | 17 | 27 | 153 | 63 | 0/260 |
| vulhunter | 0.44 | 0.80 | 0.56 | 0.62 | 0.54 | 64 | 83 | 97 | 16 | 1/260 |
| slither-0.11.3 | 0.53 | **1.00** | **0.70** | 0.73 | 0.61 | 80 | 70 | 110 | 0 | 0/260 |
| solhint-5.1.0 | 0.00 | 0.00 | 0.00 | 0.54 | 0.78 | 0 | 40 | 140 | 80 | 0/260 |
| mythril-0.24.8 | 0.52 | 0.45 | 0.48 | 0.70 | 0.82 | 36 | 33 | 147 | 44 | 0/260 |
| ethor-2023 | 0.33 | **1.00** | 0.50 | 0.44 | 0.23 | 50 | 101 | 30 | 0 | 79/260 |
| oyente | **0.75** | 0.33 | 0.46 | **0.76** | **0.95** | 21 | 7 | 134 | 42 | 4/208 |
| sailfish | 0.60 | 0.39 | 0.48 | 0.75 | 0.89 | 15 | 10 | 84 | 23 | 24/156 |
| coin-flipper | 0.30 | 0.49 | 0.37 | 0.49 | 0.49 | 39 | 91 | 89 | 41 | 0/260 |
| over-achiever | 0.31 | **1.00** | 0.47 | 0.31 | 0.00 | 80 | 180 | 0 | 0 | 0/260 |

P=Precision, R=Recall, A=Accuracy, S=Specificity, E=Errors, T=Total

Table 5.2: Ex. I - Evaluation results of three groups of tools (ML-based, conventional and baseline classifiers) on the minimal reentrancy dataset (260 contracts, ratio vulnerable:benign = 9:4)

0.4, as shown in Figure 5.1. However, as Solidity versions increase, the performance of both tools deteriorates. The performance of mando remains stable at a low level.

### 5.3.3 Results per Reentrancy Category

Our observations from the performance results split by type of reentrancy, as shown in Table 5.3, are as follows:

1. The only 10 out of 260 contracts mando classifies as benign lie in the create-based reentrancy category.
2. Mando-hgt performs best in the same-function category and worst in the delegated category.
3. In the cross-function category, vulhunter outperforms conventional tools in terms of F1.

## 5.4 Experiment II: Evaluation on a Larger Dataset with Known Ground Truth

The contracts tested in Experiment I represent minimal showcases of the reentrancy vulnerability. A real vulnerable contract deployed on-chain is typically much larger and consists of both vulnerable and non-vulnerable parts—the latter represents a source of noise we are lacking in Experiment I. Therefore, for our next experiment, we use



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

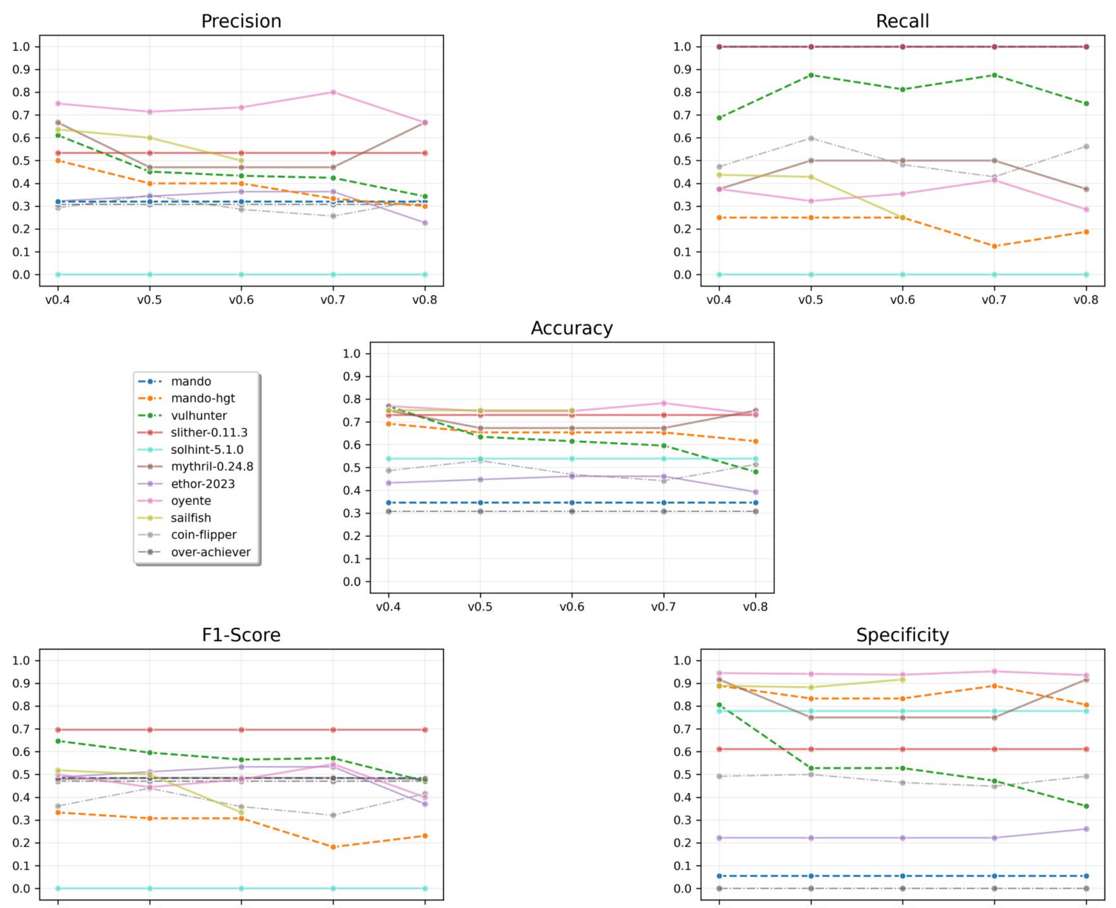



Figure 5.1: Ex. I - Performance evolution over Solidity versions

*Consolidated Ground Truth (CGT)*, a dataset constructed from previously published benchmark sets that were manually classified by the respective authors [dAS23].

We partition the full CGT Dataset into four subsets, $\mathcal{S}_1 ... \mathcal{S}_4$, as shown in Table 5.4. Because $\mathcal{S}_2$ represents the largest set compatible with our SmartBugs-based test environment, we use it as the dataset for Experiment II to evaluate the tools listed in Section 5.2.

### 5.4.1 Results based on Pre-Established GT

We extract 987 results for the subset $\mathcal{S}_4$ and determine precision, recall, F1 (all for the vulnerable class), accuracy, and specificity related to the pre-established GT of the dataset, as shown in Table 5.5. It is important to consider, that $S_4$ is class-imbalanced, with a 1:8.45 ratio of flagged (vulnerable) to non-flagged (benign) contracts. Therefore, we omit the accuracy metric when comparing it to the previous experiment on a dataset with a different (1:2.25) balance.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

| | CREATEBASED | | | | | | CROSSFUNCTION | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Tool** | **P** | **R** | **F1** | **A** | **S** | **E/T** | **P** | **R** | **F1** | **A** | **S** | **E/T** |
| mando | 0.36 | **1.00** | 0.53 | 0.46 | 0.22 | 0/65 | 0.31 | **1.00** | 0.47 | 0.31 | 0.00 | 0/65 |
| mando-hgt | 0.36 | 0.20 | 0.26 | 0.65 | 0.84 | 0/65 | 0.35 | 0.30 | 0.32 | 0.62 | 0.76 | 0/65 |
| vulhunter | 0.29 | 0.60 | 0.39 | 0.43 | 0.36 | 0/65 | 0.57 | **1.00** | **0.73** | 0.77 | 0.67 | 0/65 |
| slither-0.11.3 | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | 0/65 | 0.50 | **1.00** | 0.67 | 0.69 | 0.56 | 0/65 |
| solhint-5.1.0 | - | 0.00 | 0.00 | 0.69 | **1.00** | 0/65 | 0.00 | 0.00 | 0.00 | 0.38 | 0.56 | 0/65 |
| mythril-0.24.8 | - | 0.00 | 0.00 | 0.69 | **1.00** | 0/65 | - | 0.00 | 0.00 | 0.69 | **1.00** | 0/65 |
| ethor-2023 | 0.29 | **1.00** | 0.45 | 0.29 | 0.00 | 6/65 | 0.32 | **1.00** | 0.48 | 0.37 | 0.11 | 3/65 |
| oyente | - | 0.00 | - | 0.65 | **1.00** | 33/117 | **1.00** | 0.44 | 0.62 | 0.83 | **1.00** | 0/117 |
| sailfish | 0.60 | 0.50 | 0.55 | 0.71 | 0.83 | 4/39 | **1.00** | 0.50 | 0.67 | **0.87** | **1.00** | 8/39 |
| coin-flipper | 0.16 | 0.25 | 0.20 | 0.37 | 0.42 | 0/65 | 0.27 | 0.50 | 0.35 | 0.43 | 0.40 | 0/65 |
| over-achiever | 0.31 | **1.00** | 0.47 | 0.31 | 0.00 | 0/65 | 0.31 | **1.00** | 0.47 | 0.31 | 0.00 | 0/65 |

| | DELEGATED | | | | | | SAMEFUNCTION | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Tool** | **P** | **R** | **F1** | **A** | **S** | **E/T** | **P** | **R** | **F1** | **A** | **S** | **E/T** |
| mando | 0.31 | **1.00** | 0.47 | 0.31 | 0.00 | 0/65 | 0.31 | **1.00** | 0.47 | 0.31 | 0.00 | 0/65 |
| mando-hgt | 0.12 | 0.05 | 0.07 | 0.60 | 0.84 | 0/65 | 0.75 | 0.30 | 0.43 | 0.75 | 0.96 | 0/65 |
| vulhunter | 0.35 | 0.65 | 0.46 | 0.52 | 0.47 | 1/65 | 0.56 | 0.95 | 0.70 | 0.75 | 0.67 | 0/65 |
| slither-0.11.3 | 0.40 | **1.00** | 0.57 | 0.54 | 0.33 | 0/65 | 0.50 | **1.00** | 0.67 | 0.69 | 0.56 | 0/65 |
| solhint-5.1.0 | - | 0.00 | 0.00 | **0.69** | **1.00** | 0/65 | 0.00 | 0.00 | 0.00 | 0.38 | 0.56 | 0/65 |
| mythril-0.24.8 | **0.47** | 0.80 | **0.59** | 0.66 | 0.60 | 0/65 | 0.57 | **1.00** | **0.73** | 0.77 | 0.67 | 0/65 |
| ethor-2023 | - | - | - | - | - | 65/65 | 0.43 | **1.00** | 0.60 | 0.67 | 0.56 | 5/65 |
| oyente | 0.47 | 0.42 | 0.44 | 0.68 | 0.79 | 0/117 | **1.00** | 0.50 | 0.67 | **0.85** | **1.00** | 0/117 |
| sailfish | 0.25 | 0.20 | 0.22 | 0.60 | 0.76 | 4/39 | **1.00** | 0.38 | 0.55 | 0.84 | **1.00** | 8/39 |
| coin-flipper | 0.22 | 0.30 | 0.26 | 0.46 | 0.53 | 0/65 | 0.38 | 0.50 | 0.43 | 0.60 | 0.64 | 0/65 |
| over-achiever | 0.31 | **1.00** | 0.47 | 0.31 | 0.00 | 0/65 | 0.31 | **1.00** | 0.47 | 0.31 | 0.00 | 0/65 |

P=Precision, R=Recall, A=Accuracy, S=Specificity, E=Errors, T=Total.

Table 5.3: Ex. I - Performance per reentrancy category

| **Inclusion Criteria** | **Excluded** | **Included** |
|---|---|---|
| $\mathcal{U}$: .sol available (Total number of contracts) | - | 3,103 |
| $\mathcal{S}_1$: Pragma included | 533 | 2,570 |
| $\mathcal{S}_2$: Solidity version supported by SmartBugs ($\geq$ 0.4.11) | 34 | 2,536 |
| $\mathcal{S}_3$: GT for reentrancy available | 1,500 | 1,036 |
| $\mathcal{S}_4$: GT not conflicting | 49 | 987 |

Table 5.4: Ex. II - Number of contracts excluded and included for each subset of CGT

We observe the same behavior for mando as in our previous experiment: it classifies the vast majority of contracts as vulnerable, resulting in a high number of false positives. Mando-hgt shows improved recall ($0.21 \rightarrow 0.35$) but reduced precision ($0.39 \rightarrow 0.14$), F1 ($0.27 \rightarrow 0.20$), and specificity ($0.85 \rightarrow 0.68$) compared with our previous experiment testing minimal reentrancy showcases. It still cannot outperform the baseline dummy classifiers in terms of recall and F1 score.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

| Tool | P | R | F1 | A | S | TP | FP | TN | FN | E/T |
|---|---|---|---|---|---|---|---|---|---|---|
| mando | 0.13 | 0.97 | 0.24 | 0.17 | 0.04 | 110 | 710 | 33 | 3 | 131 / 987 |
| mando-hgt | 0.14 | 0.35 | 0.20 | 0.64 | 0.68 | 39 | 236 | 507 | 74 | 131 / 987 |
| vulhunter | 0.37 | 0.55 | 0.44 | 0.83 | 0.87 | 64 | 110 | 732 | 52 | 29 / 987 |
| vulhunter-verify | **0.40** | 0.48 | 0.44 | **0.86** | **0.91** | 48 | 72 | 732 | 52 | 83 / 987 |
| slither-0.11.3 | 0.20 | 0.93 | 0.33 | 0.55 | 0.50 | 104 | 419 | 415 | 8 | 41 / 987 |
| solhint-5.1.0 | 0.22 | 0.45 | 0.29 | 0.73 | 0.77 | 56 | 199 | 654 | 69 | 9 / 987 |
| mythril-0.24.8 | 0.36 | 0.80 | **0.50** | 0.82 | 0.83 | 65 | 114 | 539 | 16 | 253 / 987 |
| majority vote* | 0.25 | 0.87 | 0.39 | 0.66 | 0.63 | 109 | 323 | 539 | 16 | 0 / 987 |
| coin-flipper | 0.13 | 0.52 | 0.21 | 0.50 | 0.50 | 65 | 434 | 428 | 60 | 0 / 987 |
| over-achiever | 0.13 | **1.00** | 0.22 | 0.13 | 0.00 | 125 | 862 | 0 | 0 | 0 / 987 |

P=Precision, R=Recall, A=Accuracy, S=Specificity, E=Errors, T=Total
* of successful scans among slither, solhint and mythril

Table 5.5: Ex. II - Evaluation results on dataset $\mathcal{S}_4$ applying ground truth from CGT

Vulhunter achieved an improved specificity (0.54 → 0.87) but reduced precision (0.44 → 0.37), recall (0.80 → 0.55), and F1 (0.56 → 0.44). Conventional tools like Mythril showed increased recall (0.38 → 0.80) but a reduced precision (0.50 → 0.36).

When running vulhunter with active `verify` flag—a feature intended to reduce false positives—we observe that the number of errors increases compared to normal operation. Closer examination reveals that these errors occurred due to timeouts, when the tool becomes stuck in symbolic execution using the Manticore library on certain contracts for over one hour.

Comparing the performance of tested ML-based and conventional tools, we find that the pretrained vulhunter outperforms both models of mando across all metrics, and even surpasses conventional tools in terms of precision, accuracy and specificity. However, it lags behind Slither and Mythril in identifying the maximum number of vulnerable contracts, as reflected in its relatively low recall. This weakness was not observed in Experiment I, suggesting that additional noise present in larger contracts may negatively affect detection performance.

### 5.4.2 Results Based on GT Set by Conventional Tools

By using $\mathcal{S}_4$ in our experiment so far, we are ignoring the larger set of 1500 potentially testable contracts $\mathcal{S}_2 \setminus \mathcal{S}_4$. In this case, no pre-established ground truth is available to base our evaluation on. A popular method to fill this gap is majority voting of conventional detection tools (e.g. [YKK22], [SQLH22]).

Majority voting is not an ideal method as shown by the accuracy score of the method in Table 5.5. Only 66% of contracts in $\mathcal{S}_4$ are classified correctly by majority vote of conventional tools. This should be considered on studying the results of our evaluation on



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

$\mathcal{S}_2$ as shown in 5.6 where GT was established by a majority vote among Slither, Mythril, and Solhint.[14]

Compared to the previous experiments, vulhunter's recall decreases further—falling below the coin-flipper baseline classifier and mando-hgt—while its precision increases substantially. Other metrics remain in similar ranges.

| Tool | P | R | F1 | A | S | TP | FP | TN | FN | E/T |
|---|---|---|---|---|---|---|---|---|---|---|
| mando | 0.34 | 0.99 | 0.51 | 0.36 | 0.05 | 741 | 1435 | 73 | 9 | 278 / 2536 |
| mando-hgt | 0.39 | 0.36 | 0.37 | 0.60 | 0.72 | 270 | 426 | 1082 | 480 | 278 / 2536 |
| vulhunter | 0.63 | 0.25 | 0.36 | 0.69 | 0.92 | 210 | 123 | 1478 | 623 | 102 / 2536 |
| vulhunter-verify | **0.65** | 0.20 | 0.31 | **0.70** | **0.95** | 157 | 85 | 1479 | 623 | 192 / 2536 |
| coin-flipper | 0.36 | 0.51 | 0.42 | 0.50 | 0.50 | 454 | 817 | 826 | 439 | 0 / 2536 |
| over-achiever | 0.35 | **1.00** | **0.52** | 0.35 | 0.00 | 893 | 1643 | 0 | 0 | 0 / 2536 |

P=Precision, R=Recall, A=Accuracy, S=Specificity, E=Errors, T=Total

Table 5.6: Ex. II - Evaluation results on dataset $\mathcal{S}_2$ applying ground truth from majority vote of conventional tools

### 5.4.3 Tool Agreement

To evaluate the extent of agreement among the tested tools, we calculate their *overlap* of positive predictions for contracts of $\mathcal{S}_2$—a measure provided by di Angelo et al. [dADFS24, p.39]—and show the results in Table 5.7. We simplify the definition of *overlap* for the detection of a single weakness:

$$\text{overlap}(t_1, t_2) = \frac{|\text{Flagged}(t_1) \cap \text{Flagged}(t_2)|}{|\text{Flagged}(t_1)|}$$

| $t_1 \downarrow t_2 \rightarrow$ | mando | mando-hgt | vulhunter | vulhunter-v. | slither | solhint | mythril |
|---|---|---|---|---|---|---|---|
| mando | **100.0** | 31.2 | 12.6 | 9.6 | 40.8 | 21.8 | 28.9 |
| mando-hgt | 97.6 | **100.0** | 14.1 | 11.0 | 46.5 | 28.4 | 30.8 |
| vulhunter | 97.9 | 34.9 | **100.0** | 99.2 | 75.2 | 33.4 | 36.2 |
| vulhunter-verify | 97.6 | 35.7 | 100.0 | **100.0** | 77.0 | 36.8 | 37.4 |
| slither | 98.7 | 36.1 | 23.9 | 18.9 | **100.0** | 49.1 | 27.6 |
| solhint | 99.6 | 41.5 | 20.6 | 17.2 | 96.9 | **100.0** | 20.7 |
| mythril | 98.4 | 32.3 | 17.0 | 13.5 | 39.0 | 16.4 | **100.0** |

Table 5.7: Ex. II - Overlap in detected vulnerabilities [%]

Since mando classifies nearly all contracts as vulnerable, the other tools exhibit a high overlap with it in the vulnerable contracts they identify. Mando-hgt shows no significant

[14] In case of a tie, which can still occur if an analysis fails, we assume the contract is vulnerable.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

overlap with other tools. Vulhunter displays a high overlap with Slither, suggesting that the labels used by its authors for training may have been strongly influenced by Slither's results.

## 5.5 Experiment III: Retraining the Models on the Same Dataset

### 5.5.1 Dataset Construction

We combine the minimal dataset from Experiment I and $\mathcal{S}_4$ from Experiment II to retrain VulHunter and MANDO-HGT, enabling a direct comparison of their performance while avoiding potential biases arising from differences in their original training datasets.

We previously established that $\mathcal{S}_4$ is class-imbalanced with a 1:8.45 ratio of vulnerable to benign contracts. For training, we require a more balanced dataset and therefore set a target ratio of 1:2, consistent with the training dataset originally used to train VulHunter.

To achieve the target ratio, we include all 260 contracts of the minimal dataset, add all contracts from $\mathcal{S}_4$, and finally supplement the set with randomly selected benign contracts from $\mathcal{S}$ until the desired ratio is reached. Contracts with compilation errors are excluded. The resulting dataset is split into training and test sets in a 4:1 ratio, maintaining the class ratio in both subsets. Another condition we apply is that all contracts from the minimal dataset are included in the training set. The dataset statistics are shown in Table 5.8.

| Dataset | Vulnerable | Benign | Total |
|---|---|---|---|
| $\mathcal{R}_{train}$ | 157 | 314 | 471 |
| $\mathcal{R}_{test}$ | 40 | 80 | 120 |
| $\mathcal{R}$ | 197 | 394 | 591 |

Table 5.8: Ex. III - Number of contracts in dataset $\mathcal{R}$ for retraining models

### 5.5.2 Training VulHunter

The prerequisites of VulHunter's model training are instances of bytecode sequences of the dataset's contracts and the corresponding ground truth labels. The former can be extracted by executing VulHunter's `main.py` module with the `train-contracts` flag pointing to a directory containing contracts and the `train-solcversions` flag pointing to a JSON file specifying the Solidity version for each contract. The instance generator outputs a JSON file, which serves as input for model training.

To start model training we execute the following command:

```
python3 main/main.py --train-labels input/cgt_final_labels.json
  --contract-instances input/cgt_final_instances.json
```



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
--model-dir models_train --detectors reentrancy-eth
```

Model training stores the final model `torch_reentrancy-eth.pth` in the model directory provided in the training command. We export it in an updated SmartBugs compatible docker image for final evaluation on $\mathcal{R}_{test}$.

Instructions to reproduce the experiment are provided in our fork of VulHunter on GitHub[15].

### 5.5.3 Training MANDO-HGT

MANDO-HGT offers two modes for training the model: *Graph Classification* for coarse-grained detection and *Node Classification* for fine-grained detection. We select the former, as only coarse-grained labels are available for our dataset.

For model training, MANDO-HGT requires a set of extracted graphs—either *Call Graphs (CG)*, *Control Flow Graphs (CFG)*, or a fusion of both into a *heterogeneous control-flow graph (HCFG)*. We opt for HCFGs, as they were also chosen in the authors' evaluation.

The original graph extractor scripts provided by the authors require fine-grained ground truth labels—i.e., a set of vulnerable lines for each Solidity contract—regardless of the granularity chosen for in model training. Because such labels are unavailable in our case, we adapt the graph generator scripts to omit adding the fine-grained labels to the node metadata. This metadata is used only to initialize embeddings and does not serve as the ground-truth for loss calculation; those labels are provided separately to model training. Consequently, we do not anticipate a substantial negative impact on model training, although we cannot entirely rule out such effects.

After our modifications, we execute the following scripts to generate the HCFGs, which are output as `.pickle` files:

1. `call_graph_generator.py`
2. `control_flow_graph_generator.py`
3. `combination_call_graph_and_control_flow_graph_helper.py`

Also, we adapt the model training script `graph_classification.py` as follows:

1. Set the number of epochs to 50, matching the value used for training VulHunter
2. Modified the scoring procedure to run once after each epoch rather than after every contract, and to measure binary metrics for the vulnerable class instead of micro and macro scores for precision, recall, and F1, matching the behavior of our SmartBugs-based test environment.

[15] `https://github.com/stephan-klein/VulHunter`



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3. Removed cross-validation and instead train on the full provided dataset
4. Re-enabled previously commented code to allow independent testing of a trained model with a specified holdout set.

The full set of changes and instructions to reproduce the experiment are provided in our fork of Mando-HGT on GitHub[16].

Finally we train the model by executing the following command:

```
graph_classifier.py -ld ./logs/graph_classification/cfg_cg/
   node_features --output_models ./models/graph_classification/
   cfg_cg/node_features3 --dataset ./experiments/ge-sc-data/
   source_code/reentrancy/cgt_split/train --compressed_graph ./
   experiments/ge-sc-data/source_code/reentrancy/cgt/
   cfg_cg_compressed_graphs.gpickle --label ./experiments/ge-sc-
   data/source_code/reentrancy/cgt_split/mando_labels.json --
   node_feature nodetype --seed 1
```

The `node_feature` parameter specifies the type of features used as initial embeddings for the model. Some options require a separate feature extractor, which is not available to us. In practice, this choice has little impact on performance—the original authors report that results across different node features vary by no more than 1.5 percentage points for source code–based graph classification [NNX+23, Table II].

For evaluation, the MANDO inference app requires both a graph-classification model and a node-classification model to test smart contracts. Since we only train the former, we cannot evaluate performance on $\mathcal{R}_{test}$ using the MANDO inference app in our SmartBugs-based test environment. Instead, we execute the test mode of `graph_classifier.py`, ensuring that all metrics are calculated in the same manner as in our test environment:

```
graph_classifier.py --testset ./experiments/ge-sc-data/
   source_code/reentrancy/cgt_split/test --compressed_graph ./
   experiments/ge-sc-data/source_code/reentrancy/cgt/
   cfg_cg_compressed_graphs.gpickle --label ./experiments/ge-sc-
   data/source_code/reentrancy/cgt_split/test/mando_labels.json
    --node_feature nodetype --seed 1 --test
```

### 5.5.4 Results

Table 5.9 summarizes the performance of the retrained models, the original models, conventional tools, and the baseline dummy classifiers on $\mathcal{R}_{test}$. When comparing the retrained models with the original ones, we see an improvement for MANDO-HGT (M) as well as VulHunter (V): precision increases by 0.36 (M) and 0.14 (V), recall by 0.27

[16] https://github.com/stephan-klein/ge-sc-transformer



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

(V), accuracy by 0.18 (M) and 0.1 (V), and specificity by 0.27 (M). Only recall for MANDO-HGT and specificity for VulHunter remain unchanged.

In a direct comparison between the retrained architectures, *vulhunter-re* performs considerably better (precision +0.10, recall +0.40, F1 +0.32, accuracy +0.13, specificity -0.01) than *mando-hgt-re* in our setup based on coarse-grained labels. In Experiment II, the original vulhunter already outperformed conventional tools in terms of precision, accuracy, and specificity. The retrained version further increases recall, thereby boosting F1—the harmonic mean of precision and recall—to the highest value among all evaluated tools. Nonetheless, its recall still lags behind that of conventional tools such as Slither and Mythril.

| Tool | P | R | F1 | A | S | TP | FP | TN | FN | E/T |
|---|---|---|---|---|---|---|---|---|---|---|
| mando-hgt-re | 0.72 | 0.33 | 0.45 | 0.73 | **0.94** | 13 | 5 | 75 | 27 | 0 / 120 |
| mando-hgt | 0.36 | 0.34 | 0.35 | 0.56 | 0.67 | 13 | 23 | 47 | 25 | 12 / 120 |
| mando | 0.35 | 0.92 | 0.51 | 0.37 | 0.07 | 35 | 65 | 5 | 3 | 12 / 120 |
| vulhunter-re | **0.82** | 0.73 | **0.77** | **0.86** | 0.93 | 27 | 6 | 74 | 10 | 3 / 120 |
| vulhunter | 0.68 | 0.46 | 0.55 | 0.76 | 0.90 | 17 | 8 | 72 | 20 | 3 / 120 |
| slither-0.11.3 | 0.44 | 0.92 | 0.59 | 0.60 | 0.45 | 34 | 44 | 36 | 3 | 3 / 120 |
| mythril-0.24.8 | 0.59 | 0.96 | 0.73 | 0.79 | 0.72 | 27 | 19 | 50 | 1 | 23 / 120 |
| solhint-5.1.0 | 0.55 | 0.41 | 0.47 | 0.70 | 0.84 | 16 | 13 | 67 | 23 | 1 / 120 |
| coin-flipper | 0.34 | 0.50 | 0.41 | 0.52 | 0.53 | 20 | 38 | 42 | 20 | 0 / 120 |
| over-achiever | 0.33 | **1.00** | 0.50 | 0.33 | 0.00 | 40 | 80 | 0 | 0 | 0 / 120 |

*Note:* Retrained models are denoted with suffix '-re'
P=Precision, R=Recall, A=Accuracy, S=Specificity, E=Errors, T=Total

Table 5.9: Ex. III - Evaluation results of retrained models (denoted with suffix '-re'), original models and conventional tools on dataset $\mathcal{R}_{test}$



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

CHAPTER 6

# Conclusion

We began our work with the observation that recent computer science literature shows a clear trend towards AI in general, and ML in particular. Our systematic literature review confirmed that this trend is also present in publications within our domain of interest, smart contract vulnerability detection.

From the studies identified, we extracted the state of the art, with a particular focus on the ML models employed and the resulting bubble chart shown in Figure 3.3 answers our related RQ1 at a glance. It shows that most high-quality research in the last years adopts intricate deep neural networks, either processing graphs directly with GNNs or modeling sequential information derived from graphs using advanced Recurrent Neural Networks such as Bi-LSTMs or GRUs.

Based on this conclusion, for our in-depth analysis, we selected two publications: MANDO employing different versions of a GNN and VulHunter employing a Bi-LSTM. As expected, comparing these complex model architectures in a meaningful way proved challenging. Our approach—breaking down the models into their individual layers and documenting the input and output of each, while focusing on the data flow between components in a forward pass—offers insights that were absent from previous comparative work.

Besides architecture, our interest also extended to dataset selection, labeling, and pre-processing, as these are critical stages in any ML setup. In this context, we highlight the problem of missing fine-grained ground truth labels for our classification task. The authors of selected implementations adopt different strategies to address this issue, such as supplying labels from conventional tools or mitigating the effect of label noise during training, as discussed in Section 4.4. However, they often remain vague about the original source of labels or the exact labeling strategies. Also, any mitigation applied in model training, cannot completely eliminate the underlying problem in ground truth data. In an ML context, the training dataset should be both extensive and of high



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

quality—particularly with respect to ground truth labels—yet these requirements are not ideally met in the datasets used by selected publications.

In our evaluation, we were able to demonstrate the strengths and weaknesses of both models, compared to each other and to conventional tools. Overall, VulHunter achieved competitive results in all experiments. Notably, when retrained on a larger and higher-quality dataset, it even outperformed conventional tools in key metrics such as precision and F1.

For the MANDO base model, on the other hand, our experiments revealed a fundamental issue: it classified nearly all contracts as vulnerable. This behavior led to a near-perfect recall but very low precision. Since the authors reported only F1 metrics in their evaluation, we cannot contextualize this issue further, nor determine whether it was already present in their original experiments or arose from our dataset selection or test environment.

We did not observe this issue with MANDO-HGT in our tests. However, our results still revealed a considerable performance gap: while the authors of MANDO-HGT reported an F1 of 0.95, the best score we obtained was only 0.45 in Experiment III—a value within the range of our dummy baseline classifiers. In our experiments, we used the supplied models as published in MANDO's *ge-sc-machine* repository and cannot rule out that the model variant evaluated in the original publication differs from the one available in the repository—i.e., the authors may not have published subsequent improvements addressing the issues we observed.

In conclusion, further experiments are needed to determine the cause of these issues and to explain the observed performance gaps. To maximize the effectiveness of any ML-based method, future work could focus on providing fine-grained, manually validated labels for a large number of real-world contracts, based on a systematic and transparent process. Such a *gold standard* dataset could then be used to retrain ML models with minimal noise, and potentially across multiple types of vulnerabilities. Based on the positive evaluation results of VulHunter, such models might even outperform conventional tools across all metrics.

The dataset used in our evaluation, created by combining two sources of ground truth, represents a step in the right direction toward such a dataset in terms of size as well as transparency of data and label origin. However, it still lacks fine-grained labels and a homogeneous procedure for determining ground truth.

Another aspect of applying ML for smart contract vulnerability detection that we did not investigate is the performance of *Large Language Models* (LLMs) on this task. LLMs are trained on vast and diverse datasets based on public sources and, in the process, acquire knowledge about smart contract vulnerabilities. As already noted in Section 1.4, Wei et al. [WSZ+24] report promising results evaluating *GPT-4o* and *Llama-3.1-8b*.

Since then, additional models have been released, some of them specialized in code-related tasks. An integration of various LLMs into SmartBugs could enable experiments



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

comparing general-purpose ML approaches, such as LLMs, with specialized ML models like those selected in our study. Such experiments could help determine whether the costly process of training supervised ML models—including the creation of high-quality labels—remains worthwhile in the presence of large open-source language models.

Another promising direction, mentioned in Section 3.3.5, is the combination of general-purpose LLMs with supervised ML through transfer learning. Bucher et al. [BM24] demonstrate that smaller BERT-style LLMs can be fine-tuned for text classification, outperforming larger base models.

A comprehensive evaluation that directly compares supervised learning, unsupervised learning with LLMs, and transfer learning through fine-tuning LLMs for smart contract classification would represent a valuable direction for future research building on our work.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

# Appendix

## SLR Queries

To complement the information in Table 2.2, we provide the full queries $Q1 ... Q5$ executed as part of our SLR in Listings 6.1 - 6.6.

```
Abstract:(Learning OR "Artificial Intelligence" OR "Artificial-
    Intelligence" OR "AI" OR neural OR embedding OR supervised
    OR unsupervised) AND Abstract:("smart contract" OR "smart
    contracts" OR "smart-contract" OR "Smart-Contracts" OR
    Ethereum) AND AllField:(reentrancy OR "re-entrancy" OR
    reentrance OR "re-entrance") AND AllField:(github OR zenodo)
```

Listing 6.1: Q1 (ACM)

```
Title:(Learning OR "Artificial Intelligence" OR "Artificial-
    Intelligence" OR "AI" OR neural OR embedding OR supervised
    OR unsupervised) AND Title:("smart contract" OR "smart
    contracts" OR "smart-contract" OR "Smart-Contracts" OR
    Ethereum) AND AllField:("reentrancy" OR "re-entrancy" OR
    reentrance OR "re-entrance") AND AllField:(github OR zenodo)
```

Listing 6.2: Q2 (ACM)

```
(("Abstract":"learning" OR "Abstract":"artificial intelligence"
     OR "Abstract":"artificial-intelligence" OR "Abstract":"ai"
    OR "Abstract":"neural" OR "Abstract":"embedding" OR "
    Abstract":"supervised" OR "Abstract":"unsupervised") AND ("
    Abstract":"smart contract" OR "Abstract":"smart contracts"
    OR "Abstract":"smart-contract" OR "Abstract":"smart-
    contracts" OR "Abstract":"ethereum") AND ("Full Text &
    Metadata":"reentrancy" OR "Full Text & Metadata":"re-
    entrancy" OR "Full Text & Metadata":"reentrance" OR "Full
    Text & Metadata":"re-entrance") AND ("Full Text & Metadata
    ":"github" OR "Full Text & Metadata":"zenodo"))
```

Listing 6.3: Q3 (IEEE)



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
(("Document Title":"learning" OR "Document Title":"artificial
   intelligence" OR "Document Title":"artificial-intelligence"
   OR "Document Title":"ai" OR "Document Title":"neural" OR "
   Document Title":"embedding" OR "Document Title":"supervised"
    OR "Document Title":"unsupervised") AND ("Document Title":"
   smart contract" OR "Document Title":"smart contracts" OR "
   Document Title":"smart-contract" OR "Document Title":"smart-
   contracts" OR "Document Title":"ethereum") AND ("Full Text &
    Metadata":"reentrancy" OR "Full Text & Metadata":"re-
   entrancy" OR "Full Text & Metadata":"reentrance" OR "Full
   Text & Metadata":"re-entrance") AND ("Full Text & Metadata
   ":"github" OR "Full Text & Metadata":"zenodo"))
```

Listing 6.4: Q4 (IEEE)

```
"reentrancy" OR "re-entrancy" OR "reentrance" OR "re-entrance"
   AND github OR zenodo
```

Listing 6.5: Q5 (ScienceDirect)

```
allintitle: smart contract learning OR "artificial intelligence
   " OR ai OR neural OR embedding OR supervised OR unsupervised
```

Listing 6.6: Q6 (Google Scholar)

## Inference Reports

We executed MANDO's inference application and VulHunter's inference command line tool to create a report from both tools for an exemplatory vulnerable contract as shown in Figures 1 2.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

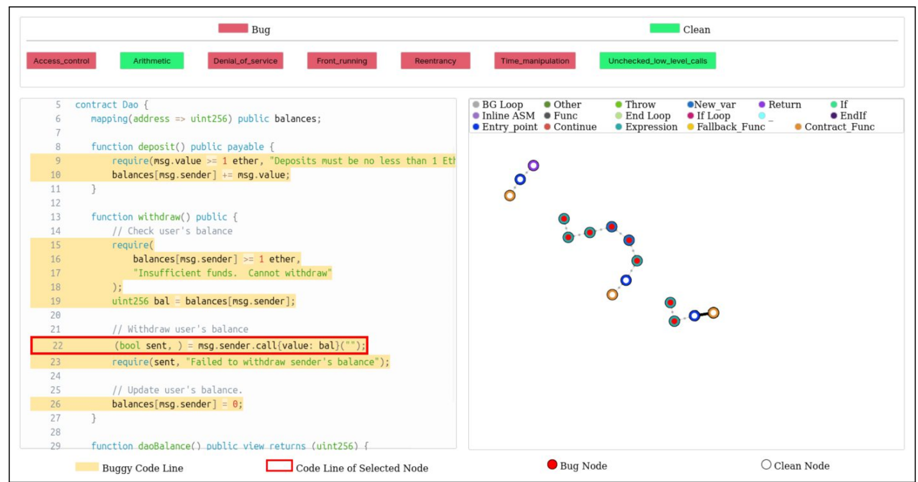
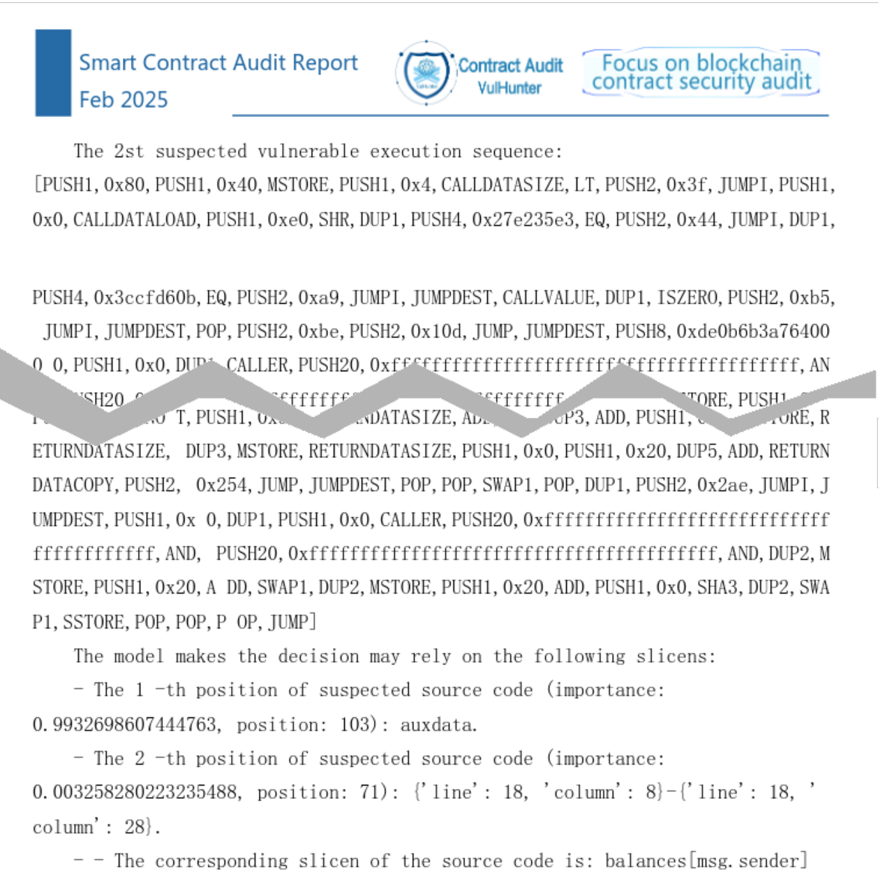



Figure 1: Mando Inference App Result


Smart Contract Audit Report
Feb 2025
Contract Audit VulHunter
Focus on blockchain contract security audit

The 2st suspected vulnerable execution sequence:
[PUSH1, 0x80, PUSH1, 0x40, MSTORE, PUSH1, 0x4, CALLDATASIZE, LT, PUSH2, 0x3f, JUMPI, PUSH1, 0x0, CALLDATALOAD, PUSH1, 0xe0, SHR, DUP1, PUSH4, 0x27e235e3, EQ, PUSH2, 0x44, JUMPI, DUP1,

PUSH4, 0x3ccfd60b, EQ, PUSH2, 0xa9, JUMPI, JUMPDEST, CALLVALUE, DUP1, ISZERO, PUSH2, 0xb5, JUMPI, JUMPDEST, POP, PUSH2, 0xbe, PUSH2, 0x10d, JUMP, JUMPDEST, PUSH8, 0xde0b6b3a76400 0 0, PUSH1, 0x0, DUP[illegible] CALLER, PUSH20, 0xff[illegible]ffffffffffffffffffffff, AN [illegible] ETURNDATASIZE, DUP3, MSTORE, RETURNDATASIZE, PUSH1, 0x0, PUSH1, 0x20, DUP5, ADD, RETURNDATACOPY, PUSH2, 0x254, JUMP, JUMPDEST, POP, POP, SWAP1, POP, DUP1, PUSH2, 0x2ae, JUMPI, JUMPDEST, PUSH1, 0x 0, DUP1, PUSH1, 0x0, CALLER, PUSH20, 0xffffffffffffffffffffffffffffffffffffff, AND, PUSH20, 0xffffffffffffffffffffffffffffffffffffffff, AND, DUP2, MSTORE, PUSH1, 0x20, A DD, SWAP1, DUP2, MSTORE, PUSH1, 0x20, ADD, PUSH1, 0x0, SHA3, DUP2, SWAP1, SSTORE, POP, POP, P OP, JUMP]

The model makes the decision may rely on the following slicens:

- The 1 -th position of suspected source code (importance: 0.9932698607444763, position: 103): auxdata.

- The 2 -th position of suspected source code (importance: 0.003258280223235488, position: 71): {'line': 18, 'column': 8}-{'line': 18, 'column': 28}.

- - The corresponding slicen of the source code is: balances[msg.sender]

Figure 2: A page of VulHunter's PDF Report

# Overview of Generative AI Tools Used

I used *GPT 4o*, *o3*, and *5* as available on `chatgpt.com` and in particular the custom GPT *Grammar & Spelling* as available in the GPT store to improve grammar, spelling and academic language for single paragraphs. All generated responses were reviewed and taken over or reworked manually.

The abstract was generated with *GPT 5 Thinking*, after providing the finished text, but substantial parts were rewritten and corrected by hand.

For extracting tool results from our experiments and converting them to LaTeX tables and figures, *GitHub Copilot* with the model *Claude Sonnet 4* was used at the level of single Jupyter code cells and in some cases for creating code and auxiliary files for integration of the tools to SmartBugs. All generated code was prompted with specific requirements, manually reviewed, tested, debugged and corrected manually if necessary.

For fixing a specific complex issue of unknown origin encountered while preparing the execution of MANDO-HGT in the SmartBugs environment, *Google Jules* as available on `jules.google.com` was used to identify the issue and propose a fix.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

# List of Figures


3.1 AI trend in CS . . . . . . . . . . 9
3.2 Classification of Machine Learning Techniques [Sar21b] . . . . . . . . . . 10
3.3 Occurrence of ML models in selected publications over time . . . . . . . . 13

4.1 NN model architectures . . . . . . . . . . 26

5.1 Ex. I - Performance evolution over Solidity versions . . . . . . . . . . 36

1 Mando Inference App Result . . . . . . . . . . 49
2 A page of VulHunter's PDF Report . . . . . . . . . . 49



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

# List of Tables

2.1 Phases and key steps of SLR [Schnd] . . . . . 5
2.2 Overview of searches performed . . . . . 6
2.3 Selection Criteria . . . . . 7
2.4 Set of primary studies and their tools . . . . . 8

3.1 Taxonomy of machine-learning paradigms and models . . . . . 11
3.2 ML paradigms employed in literature for smart contract vulnerability detection 12
3.3 ML Models used for smart contract vulnerability detection . . . . . 17

4.1 Publications and their tools selected for detailed analysis and evaluation . 19
4.2 Mapping of vulnerabilities supported by MANDO and VulHunter to SWC and DASP classifications . . . . . 20
4.3 Datasets used for training selected ml tools . . . . . 21
4.4 Layers of compared NN models . . . . . 27

5.1 Evaluation results of original authors, including best model configurations for reentrancy and best competitor tool . . . . . 30
5.2 Ex. I - Evaluation results of three groups of tools (ML-based, conventional and baseline classifiers) on the minimal reentrancy dataset (260 contracts, ratio vulnerable:benign = 9:4) . . . . . 35
5.3 Ex. I - Performance per reentrancy category . . . . . 37
5.4 Ex. II - Number of contracts excluded and included for each subset of CGT 37
5.5 Ex. II - Evaluation results on dataset $\mathcal{S}_4$ applying ground truth from CGT 38
5.6 Ex. II - Evaluation results on dataset $\mathcal{S}_2$ applying ground truth from majority vote of conventional tools . . . . . 39
5.7 Ex. II - Overlap in detected vulnerabilities [%] . . . . . 39
5.8 Ex. III - Number of contracts in dataset $\mathcal{R}$ for retraining models . . . . . 40
5.9 Ex. III - Evaluation results of retrained models (denoted with suffix '-re'), original models and conventional tools on dataset $\mathcal{R}_{test}$ . . . . . 43



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

# Bibliography

[AYCO21] Nami Ashizawa, Naoto Yanai, Jason Paul Cruz, and Shingo Okamura. Eth2vec: Learning contract-wide code representations for vulnerability detection on ethereum smart contracts. BSCI '21, page 47–59, New York, NY, USA, 2021. Association for Computing Machinery.

[BM24] Martin Juan José Bucher and Marco Martini. Fine-tuned 'small' llms (still) significantly outperform zero-shot generative ai models in text classification, 2024.

[BMY+22] Ammar Battah, Mohammad Madine, Ibrar Yaqoob, Khaled Salah, Haya R. Hasan, and Raja Jayaraman. Blockchain and nfts for trusted ownership, trading, and access of ai models. *IEEE Access*, 10:112230–112249, 2022.

[CCCP21] Filippo Contro, Marco Crosara, Mariano Ceccato, and Mila Dalla Preda. Ethersolve: Computing an accurate control-flow graph from ethereum bytecode, 2021.

[Cor24] core.edu.au - ICORE Rankings Portal, December 2024. [Online; accessed 1. Dec. 2024].

[CvMG+14] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[CXL+22] Jiachi Chen, Xin Xia, David Lo, John Grundy, Xiapu Luo, and Ting Chen. Defining smart contract defects on ethereum. *IEEE Transactions on Software Engineering*, 48(1):327–345, 2022.

[CZD+23] Hanting Chu, Pengcheng Zhang, Hai Dong, Yan Xiao, Shunhui Ji, and Wenrui Li. A survey on smart contract vulnerabilities: Data sources, detection and repair. *Information and Software Technology*, 159:107221, 2023.

[dADFS23] Monika di Angelo, Thomas Durieux, João F. Ferreira, and Gernot Salzer. Smartbugs 2.0: An execution framework for weakness detection in ethereum smart contracts, 2023.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[dADFS24] Monika di Angelo, Thomas Durieux, João F. Ferreira, and Gernot Salzer. Evolution of automated weakness detection in Ethereum bytecode: a comprehensive study. *Empirical Softw. Eng.*, 29(2), 2024.

[dAS23] Monika di Angelo and Gernot Salzer. Consolidation of ground truth sets for weakness detection in smart contracts. In *Financial Cryptography and Data Security. FC 2023 International Workshops: Voting, CoDecFin, DeFi, WTSC, Bol, Brač, Croatia, May 5, 2023, Revised Selected Papers*, pages 439—455. Springer-Verlag, 2023.

[def24] DefiLlama, October 2024. [Online; accessed 27. Oct. 2024].

[ED22] F. Emmert-Streib and M. Dehmer. Taxonomy of machine learning paradigms: a data-centric perspective. *WIREs Data Mining and Knowledge Discovery*, 12, 2022.

[FCDA20] João F. Ferreira, Pedro Cruz, Thomas Durieux, and Rui Abreu. Smartbugs: A framework to analyze solidity smart contracts, 2020.

[Fü25] Lisa Fürst. Reentrancy in ethereum smart contracts: Methods and tools for detection. Diplomarbeit, Technische Universität Wien, Wien, January 2025.

[Gar23] AI in 2024: Implementation, Productivity and Regulation, December 2023. [Online; accessed 30. Nov. 2024].

[GCC+24] Hanyang Guo, Yingye Chen, Xiangping Chen, Yuan Huang, and Zibin Zheng. Smart contract code repair recommendation based on reinforcement learning and multi-metric optimization. 33(4), April 2024.

[Gee24] GeeksforGeeks. Difference Between a Bidirectional LSTM and an LSTM. *GeeksforGeeks*, August 2024.

[GJX+21] Zhipeng Gao, Lingxiao Jiang, Xin Xia, David Lo, and John Grundy. Checking smart contracts with structural code embedding. *IEEE Transactions on Software Engineering*, 47(12):2874–2891, 2021.

[GP20] Asem Ghaleb and Karthik Pattabiraman. How effective are smart contract analysis tools? evaluating smart contract static analysis tools using bug injection. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2020, page 415–427, New York, NY, USA, 2020. Association for Computing Machinery.

[GRL+21] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. Graphcodebert: Pre-training code representations with data flow, 2021.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[GYY+24] Cuifeng Gao, Wenzhang Yang, Jiaming Ye, Yinxing Xue, and Jun Sun. sguard+: Machine learning guided rule-based automated vulnerability repair on smart contracts. *ACM Trans. Softw. Eng. Methodol.*, 33(5), June 2024.

[HCSC22] Seon-Jin Hwang, Seok-Hwan Choi, Jinmyeong Shin, and Yoon-Ho Choi. Codenet: Code-targeted convolutional neural network architecture for smart contract vulnerability detection. *IEEE Access*, 10:32595–32607, 2022.

[HDWS20] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer, 2020.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[JCX+23] Fan Jiang, Kailin Chao, Jianmao Xiao, Qinghua Liu, Keyang Gu, Junyi Wu, and Yuanlong Cao. Enhancing smart-contract security through machine learning: A survey of approaches and techniques. *Electronics*, 12(9), 2023.

[JM25] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models.* 3rd edition, 2025. Online manuscript released January 12, 2025.

[KPKT24] Bharti Khemani, Shruti Patil, Ketan Kotecha, and Sudeep Tanwar. A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions. *Journal of Big Data*, 11(1):18, 2024.

[KS24] Rasoul Kiani and Victor S. Sheng. Ethereum smart contract vulnerability detection and machine learning-driven solutions: A systematic literature review. *Electronics*, 13(12), 2024.

[LBBH98] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LCO+16] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 254–269, New York, NY, USA, 2016. Association for Computing Machinery.

[LLZ+23] Zhaoxuan Li, Siqi Lu, Rui Zhang, Ziming Zhao, Rujin Liang, Rui Xue, Wenhao Li, Fan Zhang, and Sheng Gao. Vulhunter: Hunting vulnerable smart contracts at evm bytecode-level via multiple instance learning. *IEEE Transactions on Software Engineering*, 49(11):4886–4916, 2023.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

[LQW+21] Zhenguang Liu, Peng Qian, Xiang Wang, Lei Zhu, Qinming He, and Shouling Ji. Smart contract vulnerability detection: From pure neural network to interpretable graph feature and expert pattern fusion. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2751–2759. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Main Track.

[LQW+23] Zhenguang Liu, Peng Qian, Xiaoyang Wang, Yuan Zhuang, Lin Qiu, and Xun Wang. Combining graph neural networks with expert knowledge for smart contract vulnerability detection. *IEEE Transactions on Knowledge and Data Engineering*, 35(2):1296–1310, 2023.

[LWS+24] Tianxu Liu, Yanbin Wang, Jianguo Sun, Ye Tian, Yanyu Huang, Tao Xue, Peiyue Li, and Yiwei Liu. The role of transformer models in advancing blockchain technology: A systematic survey, 2024.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[MFP+25] Nestor Maslej, Loredana Fattorini, Raymond Perrault, Yolanda Gil, Vanessa Parli, Njenga Kariuki, Emily Capstick, Anka Reuel, Erik Brynjolfsson, John Etchemendy, Katrina Ligett, Terah Lyons, James Manyika, Juan Carlos Niebles, Yoav Shoham, Russell Wald, Tobi Walsh, Armin Hamrah, Lapo Santarlasci, Julia Betts Lotufo, Alexandra Rome, Andrew Shi, and Sukrut Oak. The ai index 2025 annual report, apr 2025.

[NNX+22] Hoang H. Nguyen, Nhat-Minh Nguyen, Chunyao Xie, Zahra Ahmadi, Daniel Kudendo, Thanh-Nam Doan, and Lingxiao Jiang. Mando: Multi-level heterogeneous graph embeddings for fine-grained detection of smart contract vulnerabilities. In *2022 IEEE 9th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10, 2022.

[NNX+23] Hoang H. Nguyen, Nhat-Minh Nguyen, Chunyao Xie, Zahra Ahmadi, Daniel Kudendo, Thanh-Nam Doan, and Lingxiao Jiang. Mando-hgt: Heterogeneous graph transformers for smart contract vulnerability detection. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, pages 334–346, 2023.

[QLYH23] Peng Qian, Zhenguang Liu, Yifang Yin, and Qinming He. Cross-modality mutual learning for enhancing smart contract vulnerability detection on bytecode. In *Proceedings of the ACM Web Conference 2023*, WWW '23, page 2220–2229, New York, NY, USA, 2023. Association for Computing Machinery.

[RdAS22] Heidelinde Rameder, Monika di Angelo, and Gernot Salzer. Review of automated vulnerability analysis of smart contracts on ethereum. *Frontiers in Blockchain*, 5, 2022.




Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[rek24] Rekt - Leaderboard, October 2024. [Online; accessed 27. Oct. 2024].

[RLKD18] Michael Rodler, Wenting Li, Ghassan O. Karame, and Lucas Davi. Sereum: Protecting existing smart contracts against re-entrancy attacks, 2018.

[Sar21a] Iqbal H. Sarker. Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science*, 2(6):420, 2021.

[Sar21b] Iqbal H. Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, 2(3):160, 2021.

[Schnd] Stefan Schulte. Research methods - systematic literature reviews. Lecture slides, Distributed Systems Group, TU Wien, n.d. Available from Associate Prof. Dr.-Ing. Stefan Schulte, `s.schulte@dsg.tuwien.ac.at`.

[Sci24] Scimago Journal & Country Rank, December 2024. [Online; accessed 1. Dec. 2024].

[SQLH22] Majd Soud, Ilham Qasse, Grischa Liebel, and Mohammad Hamdaqa. Automesc: Automatic framework for mining and classifying ethereum smart contract vulnerabilities and their fixes, 2022.

[SSB14] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition, 2014.

[STZ+23] Xiaobing Sun, Liangqiong Tu, Jiale Zhang, Jie Cai, Bin Li, and Yu Wang. Assbert: Active and semi-supervised bert for smart contract vulnerability detection. *Journal of Information Security and Applications*, 73:103423, 2023.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[WEF24] The rise of smart contracts and strategies for mitigating cyber and legal risks, November 2024. [Online; accessed 26. Nov. 2024].

[WSZ+24] Zhiyuan Wei, Jing Sun, Zijian Zhang, Xianhao Zhang, Xiaoxuan Yang, and Liehuang Zhu. Survey on quality assurance of smart contracts, 2024.

[WZW+21] Hongjun Wu, Zhuo Zhang, Shangwen Wang, Yan Lei, Bo Lin, Yihao Qin, Haoyu Zhang, and Xiaoguang Mao. 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)*, pages 378–389, 2021.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[XYZ+24] Yinxing Xue, Jiaming Ye, Wei Zhang, Jun Sun, Lei Ma, Haijun Wang, and Jianjun Zhao. xfuzz: Machine learning guided cross-contract fuzzing. *IEEE Transactions on Dependable and Secure Computing*, 21(2):515–529, 2024.

[YKK22] Chavhan Sujeet Yashavant, Saurabh Kumar, and Amey Karkare. Scrawld: A dataset of real world ethereum smart contracts labelled with vulnerabilities, 2022.

[ZB23] Oualid Zaazaa and Hanan El Bakkali. A systematic literature review of undiscovered vulnerabilities and tools in smart contract technology. *Journal of Intelligent Systems*, 32(1):20230038, 2023.

[ZCH+21] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications, 2021.

[ZLQ+20] Yuan Zhuang, Zhenguang Liu, Peng Qian, Qi Liu, Xiang Wang, and Qinming He. Smart contract vulnerability detection using graph neural network. In *IJCAI*, pages 3283–3290, 2020.

[ZLQ+21] Yuan Zhuang, Zhenguang Liu, Peng Qian, Qi Liu, Xiang Wang, and Qinming He. Smart contract vulnerability detection using graph neural networks. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, IJCAI'20, 2021.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub