# Classification and Vulnerability Detection of Ethereum Energy Smart Contracts

by

Bahareh Lashkari

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Software Engineering and Intelligent Systems

Department of Electrical And Computer Engineering
University of Alberta

# Abstract

Since the advent of distributed ledger technologies, they have provided diverse opportunities in a wide range of application domains. With the transition towards a more decentralized and dynamic system, the significance of blockchain-enabled smart contracts has grown in prominence. Despite their benefits, smart contracts, are not immune to errors, vulnerabilities, and security issues. There have been several notable incidents caused by smart contract security flaws, the most significant being the DAO incident, which was triggered by a reentrancy vulnerability that resulted in the unauthorized extraction of approximately $70 million in 2016. As a result, identifying and detecting smart contract vulnerabilities has become a critical challenge that must be addressed promptly to mitigate potential financial losses caused by bug exploitation. In spite of the positive association between smart contract categories and their vulnerabilities, vulnerability analysis tools do not consider violation structures and behavior patterns across different application domains. As a result, it is imperative to analyze the smart contracts feature space to gain more insights into the characteristics of the contracts deployed.

Unlike traditional contracts, smart contracts are not written in a natural language, making it difficult to determine their content. As a result, smart contract classification based on the application domain and transaction context provides greater insight into the syntactic and semantic properties of that domain. We intend to reach greater degrees of abstraction and navigate the complexities of Decentralized Applications (DApps) design specifications by determining the contract's domain. The proposed approach will help to establish the groundwork for innovative solutions for domain-

specific classification and vulnerability detection of smart contracts in the works that follow.

In a subsequent study, we perform a domain-specific evaluation of state-of-the-art vulnerability detection tools on smart contracts. It appears that the detection accuracy of the tools varies depending on the domain. This suggests that security flaws may be domain-specific. As a result, in some domains, many vulnerabilities can be overlooked by existing analytical tools. Additionally, the overall impact of a specific vulnerability can differ significantly between domains, making its mitigation priority subject to business logic. Therefore, more effort should be directed towards the reliable and accurate detection of existing and emerging vulnerabilities from a domain-specific perspective.

In our ensuing study, this finding is used to enhance the detection of vulnerabilities within smart contracts, with a specific emphasis on an energy use case. When integrated into energy systems, smart contracts can create complex temporal and sequential dependencies, which can lead to a wide range of vulnerabilities as the structure of smart contracts becomes increasingly intricate. Current efforts to identify vulnerabilities in smart contracts rely heavily on expert-defined patterns, a method considered inefficient due to its lack of adaptability. To enhance the effectiveness of current methods, we introduce a graph attention neural network model called TL-GAT, which leverages transfer learning to detect vulnerabilities in smart contracts. This framework enables developers to independently assess vulnerabilities in each domain, allowing them to accurately identify potential issues in diverse execution environments. The evaluation results confirm that transfer learning can be used to leverage existing knowledge of vulnerabilities to improve the effectiveness of generalization when addressing vulnerabilities in a specific domain, even with limited data availability.

This thesis presents profound insights and rigorously evaluated methods for identification of smart contracts developed to facilitate energy transactions in smart grids, followed by accurate detection of reentrancy vulnerabilities in energy smart contracts.

# Preface

The research presented in this thesis was conducted under the supervision of Professor Petr Musilek. This thesis is founded on three primary publications:

1. Lashkari, Bahareh, and Petr Musilek. "Detection and Analysis of Ethereum Energy Smart Contracts." MDPI, Applied Sciences 13, no. 10 (2023).

2. Lashkari, Bahareh, and Petr Musilek. "Evaluation of Smart Contract Vulnerability Analysis Tools: A Domain-Specific Perspective." MDPI, Information 14, no. 10 (2023).

3. Lashkari, Bahareh, and Petr Musilek. "Transfer Learning with Graph Neural Networks for Vulnerability Detection in Energy Smart Contracts." Elsevier, Information Sciences (2023), under review.

Throughout my Ph.D. journey, I also seized the opportunity to delve into the intricacies of consensus mechanisms, as yet another core element of distributed ledgers. This work has been outlined in the following publications that are not incorporated in this thesis:

4. Lashkari, Bahareh, and Petr Musilek. "A comprehensive review of blockchain consensus mechanisms." IEEE Access 9 (2021): 43620-43652.

5. Lashkari, Bahareh, and Petr Musilek. "Consensus Mechanisms In Proof-of-Stake for Blockchain Networks: Fundamentals, Challenges and Approaches." Institution of Engineering and Technology (2023), in print.

# Acknowledgements

I wish to convey my profound appreciation to my academic supervisor, Professor Petr Musilek, for his exceptional mentorship and unwavering support throughout my Ph.D. program. His encouragement and insight have been instrumental in shaping the quality and direction of this thesis.

My heartfelt gratitude also extends to my family for their steadfast encouragement and unwavering faith in my capabilities. Your support has served as the cornerstone of my academic journey.

# Table of Contents


**1 Introduction** **1**

1.1 Smart Contracts . . . . . 4
1.2 Domain Classification . . . . . 5
1.3 Vulnerability Detection . . . . . 6
1.4 Thesis Motivations and Objectives . . . . . 7
1.5 Thesis Outline . . . . . 12

**2 Detection and Analysis of Ethereum Energy Smart Contracts** **13**

2.1 Abstract . . . . . 13
2.2 Introduction . . . . . 14
2.3 Background . . . . . 18
2.3.1 Energy Smart Contract . . . . . 19
2.3.2 Text Classification . . . . . 20
2.4 Related Work . . . . . 21
2.5 Methodology . . . . . 23
2.5.1 Data Collection and Pre-Processing . . . . . 23
2.5.2 Building a Domain Corpus . . . . . 26
2.5.3 Embedding Layer . . . . . 28
2.5.4 Baseline Models . . . . . 29
2.6 Evaluation Results . . . . . 31
2.7 Conclusions . . . . . 36

**3 Evaluation of Smart Contract Vulnerability Analysis Tools: A Domain-Specific Perspective** **39**

3.1 Abstract . . . . . 39
3.2 Introduction . . . . . 40
3.3 Background and Motivation . . . . . 42
3.4 Vulnerability Analysis Tools . . . . . 44
3.5 Domain-Specific Perspective . . . . . 46

3.6 Analysis . . . . . 49
3.6.1 Benchmark . . . . . 50
3.6.2 Energy . . . . . 52
3.7 Discussion . . . . . 56
3.8 Conclusions . . . . . 60

**4 Transfer Learning with Graph Neural Networks for Vulnerability Detection in Energy Smart Contracts 63**
4.1 Abstract . . . . . 63
4.2 Introduction . . . . . 64
4.3 Background . . . . . 68
4.4 Methodology . . . . . 71
4.4.1 Pre-processing . . . . . 71
4.4.2 Graph Convolutional Network Model . . . . . 72
4.4.3 Graph Attention Convolutional Network Model . . . . . 74
4.4.4 GraphSAGE . . . . . 75
4.4.5 GGNN . . . . . 77
4.4.6 TL-GAT . . . . . 78
4.5 Results and Discussion . . . . . 80
4.6 Conclusion . . . . . 84

**5 Conclusion and Future Work 86**
5.1 Conclusion . . . . . 86
5.2 Future Work . . . . . 89

**Bibliography 91**

# List of Tables

2.1 Word2Vec Embeddings. . . . . 25
2.2 Class-specific metrics from the confusion matrix report. . . . . 33
2.3 Performance of the baseline algorithms on a full-feature model. . . . . 34

3.1 Analysis results on curated dataset. . . . . 51
3.1 Analysis results on curated dataset (Cont.) . . . . . 52
3.2 Slither's reentrancy detection rate. . . . . 58

4.1 Vulnerability detection performance of each method . . . . . 83

# List of Figures


2.1 The smart contracting paradigm for energy applications . . . . . 20
2.2 Architecture of the classification pipeline. . . . . 24
2.3 Graph of domain-specific terms. . . . . 24
2.4 Energy context assessment by logistic regression. . . . . 26
2.5 Dominant energy tokens. . . . . 34
2.6 Code segment analysis I . . . . . 36
2.7 Code segment analysis II . . . . . 37
2.8 Code segment analysis III . . . . . 37

3.1 Control flow graph. . . . . 49
3.2 Vulnerability analysis workflow. . . . . 50
3.3 Vulnerabilities across different domains. . . . . 53
3.4 Contract processing time. . . . . 53
3.5 Source code metrics. . . . . 54
3.6 Contract analysis timeouts. . . . . 54
3.7 Dominant vulnerabilities. . . . . 55
3.8 Undetected vulnerabilities. . . . . 55
3.9 Conkas analysis results . . . . . 59

4.1 Reentrancy exploitation instance . . . . . 70
4.2 Proposed Vulnerability Detection Workflow . . . . . 70
4.3 Transfer Learning with GAT . . . . . 80
4.4 Evaluation results . . . . . 84

# Abbreviations

**AST** Abstract Syntax Tree.

**Bi-LSTM** Bidirectional Long Short-Term Memory.

**CBO** Coupling Between Object.

**CBOW** Continuous Bag of Words.

**CE** Code Elements.

**CLOC** Comment Lines of Code.

**DAO** Decentralized Autonomous Organization.

**DApps** Decentralized Applications.

**DIT** Deeper Inheritance Tree.

**DLT** Distributed Ledger Technology.

**ER** Element Retrictions.

**EVM** Ethereum Virtual Machine.

**FN** False Negative.

**FNR** False Negative Rate.

**FP** False Positive.

**FPR** False Positive Rate.

**GAT** Graph Attention Network.

**GCN** Graph Convolutional Network.

**GGNN** Gated Graph Neural Network.

**GNN** Graph Neural Network.

**GraphSAGE** Graph Sample and Aggregated Embeddings.

**LDA** Latent Dirichlet Allocation.

**LLOC** Logical Lines of Code.

**LR** Logistic Regression.

**LSTM** Long Short-Term Memory.

**NB** Naive Bayes.

**NF** Number of Functions.

**NL** Nesting Level.

**NLP** Natural Language Processing.

**NOS** Number of Statements.

**ReLU** Rectified Linear Unit.

**RR** Relationship Restrictions.

**SLOC** Source Lines of Code.

**SSA** Static Single Assessment.

**SVM** Support Vector Machine.

**TF-IDF** Term Frequency–Inverse Document Frequency.

**TL-GAT** Transfer Learning Graph Attention Network.

**TMN** Temporal Messaging Network.

**TNR** True Negative Rate.

**TPR** True Positive Rate.

# Chapter 1

# Introduction

The emergence of ledgers can be tracked back more than thousands of years. It was followed by a conventional banking system where data records have been authenticated by a central authority. With the advent of computers, ledgers became digitized and evolved from the preceding centralized ledger banking system, mirroring what was initially carried out on paper. A few years later, the distributed ledger technology has been proposed by Satoshi Nakamoto [93] with the intention of excluding the former authoritative environments towards a verifiable structure. Distributed Ledger Technology (DLT) enabled a novel form of recording transactions using cryptography, advanced algorithms, and massive computing capacity [7].

As a digital database instance, DLT is shared between individuals with certain characteristics that not only preserve particular communication protocols but also go through an agreement procedure that leads to a shared decision exclusive to the group of individuals that operate the DLT [48]. The widespread reputation of distributed ledgers started with the advent of the bitcoin cryptocurrency that demonstrated their potency. Their dynamic nature can accelerate transactions and reduce associated expenses by eliminating the requirements for a central authority.

The increasing trend towards decentralization fuels the quest for technologies that facilitate tamper-resistant data exchange. Similarly, blockchain is based on a peer-to-peer architecture, which has empowered several core technologies including digital

signatures, smart contracts, cryptographic hashing, and consensus mechanisms. The notion behind blockchain as a digital, distributed, and decentralized data structure is the development of transaction blocks that store digital transactions without the need for a central authority. Information concerning new transactions is appended to the chain after it has been encrypted and confirmed by the majority of the participating agents. Each block is then timestamped and cryptographically linked to the former blocks as a demonstration for the sequence of recorded transactions. As a distributed database, Blockchain comprises an expanding record of transactions accompanied by the chronological order of their occurrence. It keeps the identity of the contributors anonymous by employing digital signatures [5].

With the continued development of blockchain technology in a multitude of applications, the number of smart contracts deployed in distributed ledgers has increased tremendously. In light of the increased adoption of blockchain platforms across a wide range of decentralised apps, smart contract interoperability is constantly evolving. Accordingly, looking into the identification of contracts is one of the public blockchain network's primary issues. Unless the contract developer invests in publicizing it through designated fora, the vast majority of smart contracts remain anonymous and hardly traceable without a description. With the increasing number of smart contracts, identifying the application domain of smart contracts on public blockchains is critical for detecting smart contract vulnerabilities. The vast majority of contracts are currently anonymous and difficult to trace without public exposure. Improved identification enables targeted vulnerability assessments, customised vulnerability detection tools, improved security practises, nuanced defect mitigation, and increased transparency.

As a result, it is imperative to outline a hierarchy for providing a comprehensive mapping of smart contracts that exceed the primary query services of blockchain platforms limited to contract address, block number, transaction hash, and timestamp. An important step towards performing such searches requires accurate labeling of

the contracts, which was initially performed through an inefficient manual process. As a result, a comprehensive classification model capable of automatically classifying current or recently uploaded contracts is required [144].

Another growing concern in blockchain security is the detection of smart contract vulnerabilities. Smart contract-enabled blockchain systems provide a diverse range of advantages, encompassing the evaluation of reliability and data integrity, along with the optimization of program execution efficiency. Nonetheless, smart contracts are not exempt from security defects [15]. A multitude of security vulnerabilities can resurface throughout the lifecycle of a smart contract and since the deployed code cannot be altered the security risks stemming from these vulnerabilities become more pronounced. As a result of massive financial losses caused by smart contract security breaches in previous incursions, the ecological stability of the contract layer in widely used blockchain platforms such as Ethereum has been jeopardized.

Scalability and new security vulnerabilities will emerge as the scale of the Ethereum projects advances over time [56]. Novel vulnerability detection algorithms should be created to accurately detect and assess security threats, as well as to identify unknown vulnerabilities and determine how to mitigate them [72]. In addition, DTL platforms such as Ethereum are developing significant momentum in industries such as energy. As the prospects of blockchain-enabled smart contracts towards economical and transparent energy sector is being widely recognized, only after identifying the smart contracts deployed in transactive energy systems can a comprehensive analysis of the energy smart contracts and subsequent domain specific vulnerability detection be performed. The critical security concerns within smart grid systems amplify the potential impact of reentrancy vulnerabilities, as these systems rely on distributed ledgers to effectively manage electricity distribution across the grid. Attackers could alter critical data such as consumption records or grid configurations if there is a reentrancy vulnerability in data processing or storage functions. This manipulation could lead to billing inaccuracies, grid instability, and potential safety

hazards. Manipulation of smart grid operational functions via reentrancy attacks can cause grid instability, potentially resulting in power fluctuations or grid component damage. These issues endanger both the grid infrastructure and its users. In addition, exploiting vulnerabilities in one region of the smart grid system could cause a cascade effect, impacting the interconnected systems. A breach in one area has the potential to jeopardise the grid's overall integrity. Therefore, the application-based identification of smart contracts toward a domain-specific vulnerability detection for transactive energy systems, holds significant importance.

## 1.1 Smart Contracts

In essence, smart contracts are self-executing collections of codified agreements deployed on decentralized networks of blockchain. A smart contract is a source code entity that operates within the blockchain and, once deployed, functions in adherence to a predetermined code logic [68]. Smart contracts support decentralization through automated execution, blockchain integration, and immutable protocols. They can also contribute to trust through transparency, automated execution, immutable history, and cryptography [9].

As event-driven programs, smart contracts facilitate trusted transactions, allowing anonymous parties to exchange digital assets or data [160]. The smart contract is the main pillar of Ethereum and is widely adopted in various business domains. As a proclaimed framework for integrating the execution of smart contracts, Ethereum offers great capacity in the development of Decentralized Applications (DApps). Most Ethereum smart contracts are written in Solidity, a high-level object-oriented programming language, and then compiled into bytecode for execution using the Ethereum Virtual Machine (EVM). The bytecode compilation is followed by the deployment of the code on the blockchain, resulting in an exclusive 160-bit hexadecimal hash contract address. This address serves as the unique identifier for the smart contract. Users initiate the contract code execution by sending a transaction request

to the contract address. The EVM then autonomously executes the contract, creating a level of abstraction between the executing code and the machine on which it runs, thus isolating the DApps and their corresponding hosts [120].

In recent years, much attention has been given to blockchain platforms supporting smart contracts for the development of decentralized applications. Smart contracts have been designed and deployed in a broad array of applications, from fund management to the power grid. Given that smart contracts are deployed on a blockchain that was primarily intended to store financial transactions, the most instinctive applications of smart contracts involve trading assets between contracting parties [176].

## 1.2 Domain Classification

Coupled with increasing progression toward a more decentralized and dynamic energy system, the viability of blockchain-enabled smart contracts in transactive energy systems has become prominent. Energy smart contracts are employed for automatic execution and monitoring of energy trading events [58, 69]. An energy smart contract is implemented with instructions to retrieve the capacity and price offered by generators. The offer is then consigned to the prospective buyers for the bidding process to begin. A double auction is a commonly used mechanism to settle the price. Once the price has been cleared, the grid power flow is examined to ensure the feasibility of the allocations.

Considering the multitude of deployed contracts, the lack of methodologies (such as classification models for the analysis of energy smart contracts) makes it challenging to gain insights into this ecological environment and to identify vulnerable smart contracts once deployed. Given that search for vulnerabilities can be optimized by factoring intriguing elements such as the semantics of the defects, it is important to design and employ smart contract analysis tools to gain a broader knowledge of contracts concerning their underlying domains [64], [6].

According to our experience, there is no consistent set of smart contract specifications

in published research work, and smart contracts associated with comparable practices are often classified differently. Since complete formal characterizations of a smart contracts intended behavior are rarely available, it is imperative to describe smart contracts from a domain-specific standpoint. Furthermore, addressing prevalent smart contract vulnerabilities mandates a semantic and syntactic understanding of the compromised contract.

With the increasing number of smart contracts approximately 670,000 smart contracts deployed each month [31], identifying smart contracts designed primarily for the energy domain and deployment on smart grids allows targeted vulnerability assessment, reinforced security practices, refined defect mitigation, and a higher degree of transparency [76].

## 1.3 Vulnerability Detection

The immutability and tamper resistance of blockchain frameworks are additionally enforced on smart contracts, ensuring that any terms documented in a smart contract cannot be altered once they have been published. However, smart contracts encounter security challenges, including an error-prone programming language with exploitable development bugs that are often overlooked or discovered only after deployment on the blockchain, making it difficult to fix them. Smart contracts frequently govern substantial financial assets, making them attractive and easily targeted by malicious actors seeking financial gains. Bug exploitation in smart contracts can lead to severe consequences that affect the entire blockchain ecosystem, rather than just individual contracts.

There have been several notable incidents caused by smart contract flaws, the most significant being the Decentralized Autonomous Organization incident, which was triggered by a reentrancy vulnerability that resulted in the unauthorized extraction of approximately $70 million in 2016 [175], [137]. Later, in 2018, the SpankChain [89] contract was the target of a reentrancy attack, that ended in a $40,000 misappropriation

due to insufficient authority governance. In a similar vein the Uniswap [142] and Lendf.Me [156] projects encountered this vulnerability this time in January 2020 [53]. In May 2021, the flash.sx smart contract was targeted by a reentrancy attack, resulting in the theft of approximately 1.2 million EOS and 462,000 USDT despite a prior security audit [168]. It is evident that security defects have the potential to result in irreversible consequences. Reentrancy, among the most severe vulnerabilities in smart contracts, is spotted and leveraged frequently, compromising trust in smart contract-based applications. As a result, identifying and detecting smart contract vulnerabilities has become a critical challenge that must be addressed promptly to mitigate potential financial losses caused by bug exploitation [111].

## 1.4 Thesis Motivations and Objectives

The development of secure and reliable smart contracts can be extremely challenging due to domain-specific vulnerabilities and constraints associated with various business logics. Current methods rely heavily on expert-defined patterns and have difficulty dealing with multifaceted intrusions, calling for more robust approaches. Therefore, overdependence on environment-defined parameters in the contract execution logic binds the contract to the manipulation of such parameters and is perceived as a security vulnerability. In response to the widely recognized prospects of blockchain-enabled smart contracts towards an economical and transparent energy sector, this thesis proposes a methodology for the detection, analysis and vulnerability detection of energy smart contracts.

The primary objectives of this thesis are as follows:

- *Objective 1: Underscoring the significance of categorizing contracts based on their application domain and transaction context, obtaining valuable insights into their syntactic and semantic properties, thereby enhancing their comprehension.*
  - Recognition of the challenges inherent in developing secure and dependable

smart contracts due to vulnerabilities and constraints unique to various business logics within different domains.

  - Classification of smart contracts by employing a methodology that captures the semantic and syntactic characteristics of energy smart contracts.
  - Analysis and investigation of patterns related to the distribution of code segments, the predominant use of certain elements and the recurrence of specific contracts across the Ethereum network.

- *Objective 2: Evaluation of the state-of-the-art vulnerability detection tools for smart contracts from a domain-specific viewpoint.*
  - Investigating the smart contract structures and procedures by considering logical and language-dependent features specific to diverse application domains.
  - Examination of the code embedding of energy smart contracts and the assessment of their vulnerabilities within transactive energy systems.
  - Highlighting the risk of overlooking vulnerabilities in particular domains.
  - Drawing attention to the pronounced differences in the effects of specific vulnerabilities in different domains.

- *Objective 3: Enhancing the accuracy of vulnerability detection through transfer learning and domain-specific analysis*
  - Highlighting the inefficiency and lack of adaptability in approaches that rely on expert-defined patterns.
  - Improving generalization for vulnerability detection within a specific domain, with limited data availability.
  - Facilitating precise detection of reentrancy attacks in transactive energy systems.

In pursuit of the aforementioned objectives, we embark on three research studies. The initial research, pertaining to the first objective, is primarily dedicated to recognizing the context of smart contracts and subsequently categorizing contracts within the energy domain. The second objective is fulfilled in a subsequent research phase where the focus lies on assessing cutting-edge vulnerability detection tools for smart contracts. This evaluation is conducted with a domain-specific perspective, by considering logical and language-dependent features specific to diverse application domains. The third research stage, corresponding to the third objective, showcases the capacity of transfer learning to leverage existing knowledge of vulnerabilities for the purpose of improving generalization when addressing vulnerabilities within a specific domain, with limited data availability.

The main contributions of this thesis, corresponding to the aforementioned studies, are as follows:

**Research study 1: "Detection and Analysis of Ethereum Energy Smart Contracts" [76] (Chapter 2)**

**Aim:** Smart contract classification based on the application domain and transaction context provides deeper insight into the syntactic and semantic properties of that domain. With the progression towards a more decentralized and dynamic energy system, the impact of blockchain-enabled smart contracts in transactive energy systems has gained prominence. As a result, it is imperative to analyze the energy smart contract feature space to gain more insights into the characteristics of contracts deployed for energy transactions. This study proposes an approach to discriminate energy smart contracts using the publicly accessible Ethereum source codes. Natural Language Processing (NLP) and machine learning classification algorithms are employed to detect and properly label energy smart contracts. To begin, a domain-specific embedding layer is generated to identify and analyze energy tokens and energy-related terms. Subsequently, both the energy corpus and categorical attributes are employed as baselines for training of the classification algorithms.

**Outcome:** Logistic regression, naive bayes, and support vector machine are implemented as classifiers. The classification performance of each algorithm is then evaluated using accuracy, precision, recall, and F1-score metrics. Energy smart contracts are detected with up to 98.34% accuracy, with Logistic Regression (LR) outperforming the other algorithms. Detected contracts are further examined to discern any discrepancies or patterns in the distribution of code segments, the predominant use of specific functions, and recurring contracts across the Ethereum network. When compared to non-energy smart contracts, the results obtained on the distribution of energy code segments imply that the development of energy contracts tends to prioritize the adoption of contracts and libraries over interfaces. Finally, the same analysis among functions in both classes entails the adoption of comparable functions across each category, with more prevalent adoption among energy contracts.

**Research Study 2: "Evaluation of Smart Contract Vulnerability Analysis Tools: A Domain-Specific Perspective" [77] (Chapter 3)**

**Aim:** Smart contracts in the energy domain provide the necessary versatility to consolidate diverse processes according to the requirements of the application. Despite the positive association between smart contract categories and their vulnerabilities, vulnerability analysis tools do not consider violation structures and behavior patterns across different application domains. Unfortunately, there is no perfect contract analysis tool for any and all contracts and their underlying business logic. Furthermore, the results of vulnerability analysis tools cannot be replicated in the absence of the appointed data set. To examine this gap, this study evaluates the benchmark vulnerability analysis tools on classified and curated contracts. This classification allows for an independent assessment of each domain's vulnerability source so that developers can differentiate between domain-specific vulnerabilities when paired with different execution environments.

**Outcome:** The independent assessment of vulnerability analysis tools on different domains revealed that the detection accuracy of the tools varies depending on the domain, since the benchmark tools did not demonstrate the accuracy claimed in the curated contracts. Furthermore, the overall impact of a comparable vulnerability in one application domain may be more profound and detrimental than in another. This makes the mitigation priority of these defects subject to business logic. In addition, energy contracts demonstrate above-average security flaws and take longer to process, increasing the likelihood of failure or a timeout. The evaluation results underscore the competence of symbolic execution for the analysis of energy contracts. Accordingly, analysis tools that incorporate symbolic execution outperform code transformation coupled with constraint solving in detecting reentrancy in energy contracts. Although the vulnerability analysis workflow used in this research is only applicable to smart contract source code, it is transferable to any application domain in the presence of contracts in the corresponding domain.

**Research Study 3: "Transfer Learning with Graph Neural Networks for Vulnerability Detection in Energy Smart Contracts" (Chapter 4)**

**Aim:** When integrated into energy systems, smart contracts can create complex temporal and sequential dependencies, which can lead to a wide range of vulnerabilities as the structure of smart contracts becomes increasingly intricate. Current efforts to identify vulnerabilities in smart contracts rely heavily on expert-defined patterns, a method considered inefficient due to its lack of generality and adaptability. Additionally, manual expert inspection is a time-consuming and resource-intensive process to collect sufficiently large and properly labeled datasets of smart contracts within specific domains. To enhance the effectiveness of current methods, we introduce a graph attention neural network model called Transfer Learning Graph Attention Network (TL-GAT), which leverages transfer learning to detect vulnerabilities in smart contracts. This framework enables developers to independently assess vulnerabilities in

each domain, allowing them to accurately identify potential issues in diverse execution environments.

**Outcome:** The evaluation results confirm that transfer learning can be used to leverage existing knowledge of vulnerabilities to improve the effectiveness of generalization when detecting vulnerabilities in a specific domain, even with limited data availability. We evaluated the effectiveness of the proposed model using four prominent Graph Neural Network (GNN) architectures, Gated Graph Neural Network (GGNN), Graph Attention Network (GAT), Graph Convolutional Network (GCN) and Graph Sample and Aggregated Embeddings (GraphSAGE). GAT with Transfer Learning outperformed all other models in the framework, achieving a 99.06% accuracy, 98.50% precision, 100% recall, and an F1-score of 99.25%. The obtained performance can be attributed to TL-GAT's ability to extract insights from a pre-trained source domain, which improves its overall predictive capacity. Even though the vulnerability analysis workflow utilized in this research is specifically designed for energy smart contracts and reentrancy vulnerabilities, it can be extended to other application domains or security defects in presence of labeled contracts pertaining to the desired vulnerability within target domain.

## 1.5 Thesis Outline

The subsequent sections of this thesis are organized as follows. Chapter 2, delves into detection and classification of smart contract exploring smart contracts code elements and providing valuable insights into their syntactic and semantic properties. This is followed by Chapter 3 examining smart contract vulnerability analysis tools from a domain specific perspective. Chapter 4 introduces our proposed vulnerability detection framework based on transfer learning and graph neural network. Chapter 5, concludes our contributions and offers insights into possible directions for future research.

# Chapter 2

# Detection and Analysis of Ethereum Energy Smart Contracts

## 2.1 Abstract

As blockchain technology advances, so has the deployment of smart contracts on blockchain platforms, making it exceedingly challenging for users to explicitly identify application services. Unlike traditional contracts, smart contracts are not written in a natural language, making it difficult to determine their provenance. Automatic classification of smart contracts offers blockchain users keyword-based contract queries and a streamlined effective management of smart contracts. In addition, the advancement in smart contracts is accompanied by security challenges, which are generally caused by domain-specific security breaches in smart contract implementation.

The development of secure and reliable smart contracts can be extremely challenging due to domain-specific vulnerabilities and constraints associated with various business logics. Accordingly, contract classification based on the application domain and the transaction context offers greater insight into the syntactic and semantic properties of that class. However, despite initial attempts at classifying Ethereum smart contracts, there has been no research on the identification of smart contracts deployed in transactive energy systems for energy exchange purposes.

In this article, in response to the widely recognized prospects of blockchain-enabled smart contracts towards an economical and transparent energy sector, we propose a

methodology for the detection and analysis of energy smart contracts. First, smart contracts are parsed by transforming code elements into vectors that encapsulate the semantic and syntactic characteristics of each term. This generates a corpus of annotated text as a balanced, representative collection of terms in energy contracts. The use of a domain corpus builder as an embedding layer to annotate energy smart contracts in conjunction with machine learning models results in a classification accuracy of 98.34%. Subsequently, a source code analysis scheme is applied to identified energy contracts to uncover patterns in code segment distribution, predominant adoption of certain functions, and recurring contracts across the Ethereum network.

## 2.2 Introduction

As blockchain technology advances, so has the deployment of smart contracts on blockchain platforms, making it exceedingly challenging for users to explicitly identify application services. Unlike traditional contracts, smart contracts are not written in a natural language, making it difficult to determine their provenance. Automatic classification of smart contracts offers blockchain users keyword-based contract queries and a streamlined effective management of smart contracts. In addition, the advancement in smart contracts is accompanied by security challenges, which are generally caused by domain-specific security breaches in smart contract implementation.

The development of secure and reliable smart contracts can be extremely challenging due to domain-specific vulnerabilities and constraints associated with various business logics. Accordingly, contract classification based on the application domain and the transaction context offers greater insight into the syntactic and semantic properties of that class. However, despite initial attempts at classifying Ethereum smart contracts, there has been no research on the identification of smart contracts deployed in transactive energy systems for energy exchange purposes.

In this article, in response to the widely recognized prospects of blockchain-enabled smart contracts towards an economical and transparent energy sector, we propose a

methodology for the detection and analysis of energy smart contracts. First, smart contracts are parsed by transforming code elements into vectors that encapsulate the semantic and syntactic characteristics of each term. This generates a corpus of annotated text as a balanced, representative collection of terms in energy contracts. The use of a domain corpus builder as an embedding layer to annotate energy smart contracts in conjunction with machine learning models results in a classification accuracy of 98.34%. Subsequently, a source code analysis scheme is applied to identified energy contracts to uncover patterns in code segment distribution, predominant adoption of certain functions, and recurring contracts across the Ethereum network.

The emerging trend towards decentralization leads the pursuit for technologies that facilitate tamper-proof data exchange. Similarly, blockchain is based on a peer-to-peer architecture, which has empowered several core technologies including digital signatures, smart contracts, cryptographic hashing, and consensus mechanisms [10, 63, 92]. With the continued development of blockchain technology in a multitude of applications, the number of smart contracts deployed in distributed ledgers has increased tremendously. Smart contracts [130] are self-executing decentralized applications running on blockchain used for governance of financial assets that once deployed are autonomous and immutable. Smart contracts adhere to the underlying configuration of distributed ledgers and inherit automation, immutability, and decentralization qualities [139].

With the widespread adoption of blockchain platforms across various decentralized applications, smart contract interoperability is continuously evolving. Hence, Ethereum [25] has become a prominent smart-contract-based blockchain platform due to the increasing adoption of its decentralized applications. Ethereum's surging popularity can be attributed to its high level of robustness and adaptability for a wide range of applications [101]. Ethereum currently hosts over four million smart contracts with approximately 670,000 smart contracts deployed each month and over 3000 DApps [31]. Therefore, a major concern for users is identifying the desired

application service among dozens of smart contracts in a timely and efficient manner.

The identification of contracts is one of Ethereum's primary challenges. Some smart contract developers make their source code available, along with a description of its context and purpose. Unless the contract developer invests in publicizing it through designated fora, the vast majority of smart contracts remain anonymous and hardly traceable without a description. With the increasing number of smart contracts, assisting users to identify their required service among a massive number of contracts has become an ongoing challenge. As a result, it is imperative to outline a hierarchy that provides a comprehensive mapping of smart contracts that exceed the primary query services of blockchain platforms that are limited to contract address, block number, transaction hash, and timestamp.

An important step towards performing such searches requires accurate labeling of the contracts, which was initially performed through an inefficient manual process. As a result, a comprehensive classification model capable of automatically classifying current or recently uploaded contracts is required.

Research has been carried out to help comprehend what smart contracts do and to enable contract searches based on their context and purpose. The proposed methods are centered on learning the characteristics and the structural code embedding of smart contracts. However, the existing classification and topic modeling schemes for smart contracts are limited to application domains, such as entertainment, management, the IoT, lottery, gambling, gaming, and so on [57, 127, 135]. This is why recent studies have underlined the significance of blockchain-enabled peer-to-peer energy trading systems [170] leveraging smart contracts [147].

The advent of blockchain technology offers the potential to securely automate P2P energy trading [29, 115]. Smart contracts have proven to be effective for autonomous and secure execution of end-to-end energy transactions based on local consumer preferences. Ethereum's smart contracts are regarded as strict protocols on the blockchain that allow energy transactions to be carried out once all prerequisites

have been met. Energy exchanges are monitored as financial transactions, with the corresponding resource consumption quantified in gas units and remitted in Ether at the gas price [50, 88]. However, despite the prevalent adoption of smart contracts in energy applications that streamline consumer and prosumer interactions towards a robust settlement process, no research has been conducted on the identification and analysis of energy smart contracts. Considering the multitude of deployed contracts, the lack of methodologies (such as classification models for the analysis of energy smart contracts) makes it challenging to gain insights into this ecological environment and to identify vulnerable smart contracts once deployed.

To the best of our knowledge, there has been no contribution explicitly addressing the detection and analysis of energy smart contracts. The key contributions of this article can be summarized as follows. This chapter demonstrates the significance of domain-specific classification and analysis tools for smart contracts. This is accomplished using a method that leverages contextual terms for classification of smart contracts using code and comments.

The proposed classification pipeline provides a method for the detection and labeling of energy smart contracts using their source code. It uses feature engineering methods to produce domain-specific corpora, which are subsequently embedded within machine learning classification models. A domain-specific analysis is a method of deriving technical and language-dependent features to enhance the structural and procedural understanding of smart contracts. This is achieved by employing contextual terms at a lexical level, searching for key terms and attribute tags to develop the energy corpus that will aid in deciphering the code's context and retrieving energy contracts.

By discerning the domains, we intend to reach higher levels of abstraction and handle the intricacies of DApps design specifications. The developed model captures the entire lexicon of transactions used for developing an energy smart contract. Finally, energy smart contracts are analyzed to identify patterns in the distribution of code segments, the predominance of specific functions, and recurring contracts across the

Ethereum network.

The proposed approach can be used by contract developers to track similar contracts deployed on Ethereum. Contract level representation is used to include the highest level of granularity in the classification; thus, the proposed method is applicable to any application domain by using the key terms and attributes pertaining to that domain. It develops the corresponding corpus using contextual terms, and the collected attributes are then used to train the classifier.

This methodology can be further extended to include measures for anomaly detection and malicious contract detection in the context of energy smart contract analysis. Security practitioners can use it to investigate the potential vulnerabilities of energy smart contracts. The search for vulnerabilities can be optimized by factoring intriguing elements such as the semantics of the defects while vulnerable contract's syntactic representation is protected to the greatest extent possible. This an important step towards the design and development of auditing systems to address the identified vulnerabilities.

The article is organized as follows. Section 2.3 introduces energy smart contracts and provides background information on smart contracts and natural language processing. In Section 2.4, related works are discussed to further highlight the contributions of this research. Section 2.5 describes the research methodology and the proposed classification pipeline, followed by Section 2.6 which showcases the classification results obtained by the baseline models. Finally, Section 2.7 summarizes the conclusions and future research directions.

## 2.3 Background

To ensure effective identification of energy smart contracts, this study is based on Natural Language Processing (NLP). It is important to explore the core concepts of text classification in line with fundamental principles pertaining to the definition of smart contracting for the energy domain, as well as what constitutes a generic energy

smart contract. Section 2.3.1 provides an introduction to energy smart contracts, followed by an overview of textual data classification in Section 2.3.2.

### 2.3.1 Energy Smart Contract

In recent years, much attention has been given to blockchain platforms supporting smart contracts for the development of decentralized applications. Smart contracts have been designed and deployed in a broad array of applications, from fund management to the power grid. Given that smart contracts are deployed on a blockchain that was primarily intended to store financial transactions, the most instinctive applications of smart contracts involve trading assets between contracting parties [176]. Coupled with increasing progression toward a more decentralized and dynamic energy system, the viability of blockchain-enabled smart contracts in transactive energy systems has become prominent.

Energy smart contracts are employed for automatic execution and monitoring of energy trading events [58, 69]. An energy smart contract is implemented with instructions to retrieve the capacity and price offered by generators. The offer is then consigned to the prospective buyers for the bidding process to begin. A double auction is a commonly used mechanism to settle the price. Once the price has been cleared, the grid power flow is examined to ensure the feasibility of the allocations.

According to Desen et al. [69], the information is transmitted through six different layers in the energy smart contract workflow, as illustrated in Figure 2.1. The first layer involves the transmission of input data in the form of bids, offers, voltage levels, or availability signals from any agent involved in the peer-to-peer transaction for the demand side to be automatically triggered.

The second layer implements innovative optimization and energy management algorithms to address potential inconsistencies within contracted and delivered energy. Any computationally intensive control algorithms should be placed off smart contracts to minimize extra computational expenses imposed by implementing them in this

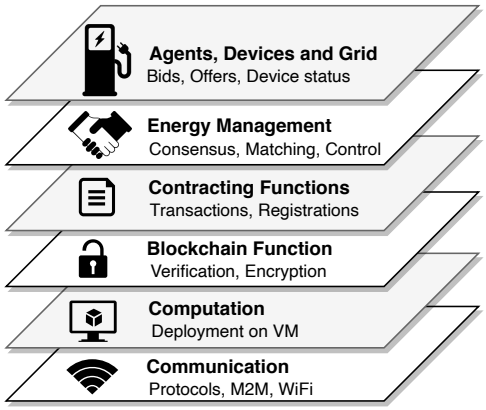


Figure 2.1: The smart contracting paradigm for energy applications

layer. The smart contract is coded in the third layer using solidity. This allows the preceding layers to incorporate different programming languages. This layer handles all financial transactions, including gas expenditure and agent registration.

Layer 4 controls the verification and encryption requirements for the smart contract's placement into the block. Layer 5 is where implementation and execution occur, which require interaction with virtual machines, namely the Etheruem Virtual Machine (EVM). Subsequently, the data are transmitted across communication protocols using wired or wireless communication means.

### 2.3.2 Text Classification

Text classification is a fundamental aspect of NLP used for topic labeling, sentiment analysis, and spam detection [51, 90]. It can be performed either through automatic labeling or manual explanation. However, dealing with an overwhelming volume of text data embodies the significance of automatic labeling approaches.

Text classification can be performed automatically using rule-based or machine learning methods. The rule-based method follows a pre-defined rule set for classification, thus necessitating a coherent understanding of the domain. Machine learning approaches, on the other hand, perform classification by analyzing the text [166]. They have proven to be effective for unstructured data. The machine-learning-based

text classification approach derives feature representations from text in conjunction with domain knowledge. The extracted features are then applied as inputs to train the classifier.

It should be noted that smart contracts are profoundly different from the standard text. Smart contracts include source code and comments, both of which convey semantic information. However, they may also carry non-descriptive or non-existent comments, resulting in semantic sparsity. This prompts the adoption of domain knowledge to reduce the semantic sparsity of smart contracts.

## 2.4 Related Work

Smart contracts, unlike conventional contracts, are not written in natural language, making it difficult to determine their context. When compared to conventional programming languages, such as C and Java, uniform understanding of Ethereum smart contracts is relatively limited.

Nonetheless, there have been a few attempts at classifying Ethereum smart contracts prior to this research. However, smart contract classification described in the existing works is not necessarily consistent with Buterin's initial classification of three tiers of financial, semi-financial, and non-financial applications [12]. For instance, Shi et al. [117] applied NLP on bytecode to classify contracts as governance, finance, gaming, wallet, and social, although wallets are a subset of financial applications, according to Buterin's early classification. Using Long Short-Term Memory (LSTM), Hu et al. [57] classified Ethereum smart contracts by identifying six behavior patterns analysing transactions including game, gambling, exchange, finance, high-risk, and social transactions. Later, Tian et al. [135] developed a smart contract classification strategy based on Bidirectional Long Short-Term Memory (Bi-LSTM) and Gaussian Latent Dirichlet Allocation (LDA) to classify contracts as entertainment, management, lottery and tools, finance, IoT, and others.

According to our experience, there is no consistent set of smart contract specifications

in published research work, and smart contracts associated with comparable practices are often classified differently.

The proposed study extracts domain models with the intended goal of deriving business logic from current Ethereum-based Dapps aiming at transactive energy systems. Complete formal characterizations of a program's intended behavior are rarely available; thus, it is imperative to describe smart contracts from a domain-specific standpoint. Furthermore, addressing prevalent smart contract vulnerabilities mandates a semantic and synthetic understanding of the compromised contract.

Although the transparent execution of smart contracts has enhanced the readability of blockchain-enabled systems, the characteristics of distributed ledgers make it extremely challenging to revoke vulnerable smart contracts once they are deployed. As a result, massive financial losses caused by smart contract security breaches in former intrusions have compromised the ecological stability of the contract layer in widely adopted blockchain platforms such as Ethereum. Hence, a growing concern in blockchain security is the detection of smart contract vulnerabilities [152, 154]. Scalability and new security vulnerabilities will emerge as the scale of the Ethereum projects advances over time. Novel vulnerability detection algorithms should detect and assess novel security threats and determine how to mitigate them. Accordingly, the search for vulnerabilities can be optimized by factoring intriguing elements such as the semantics of the defects.

In fact, contract classification based on the application domain and transaction context offers greater insight into the syntactic and semantic properties of a given class of contracts. This can be further used for the design and implementation of customized vulnerability and fault detection mechanisms for a specific domain, including transactive energy systems [47]. Consequently, it is important to design and employ smart contract analysis tools to gain a broader knowledge of contracts concerning their underlying domains [102].

This will help to establish the groundwork for the development of domain-specific

vulnerability detection algorithms for the detection and mitigation of unknown vulnerabilities and facilitates protecting vulnerable contracts' syntactic representation to the greatest extent possible.

## 2.5 Methodology

To evaluate the grammatical, symbolic, and arithmetic characteristics of smart contracts, this study analyzes their source codes using machine learning algorithms. They are deployed as predictive models for detecting Ethereum energy smart contracts.

As depicted in Figure 2.2, the proposed classification pipeline can be deconstructed into three main stages of pre-training, training, and testing. Pre-training encompasses data collection and feature engineering. This adheres to the embedding layer and baseline models under training, after which the models are tested and evaluated.

Logistic Regression (LR), Naive Bayes (NB), and Support Vector Machine (SVM) are employed as classifiers and a comparative performance analysis is carried out on the obtained results.

As shown in Figure 2.2, pre-training embeddings and domain-specific embeddings form the corpus. Its components are further illustrated in the form of the graph shown in Figure 2.3 and the embeddings listed in Table 2.1, which can be interpreted in terms of their relative frequency (see Section 2.6 for an example).

### 2.5.1 Data Collection and Pre-Processing

This study is performed on smart contracts solidity source codes retrieved from etherscan.io [31]. Since the code is explicitly used as the input for the classifier, it must be parsed to acquire an appropriate code representation form [91, 143]. Code representation is performed using a customized solidity parser at the contract level, function level, comment level, and token level to identify semantic information within the focal points. As a result, contract level representation is employed to include

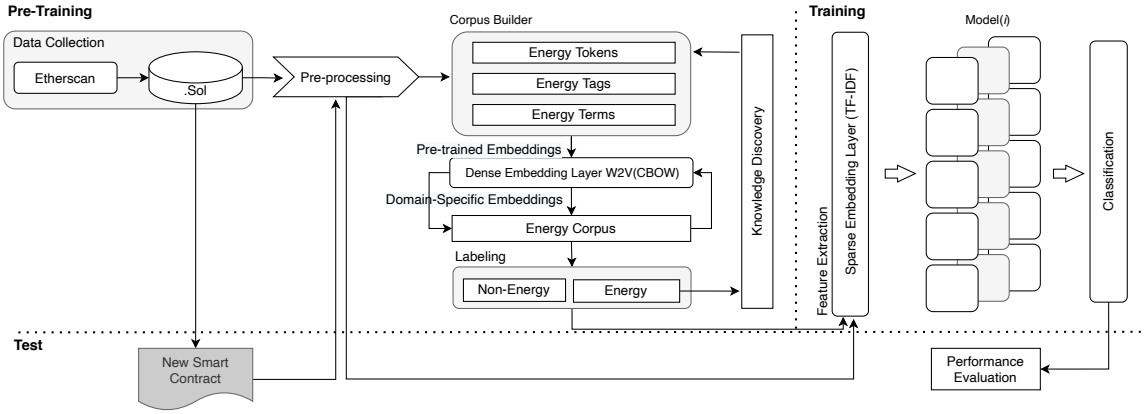


Figure 2.2: Architecture of the classification pipeline.

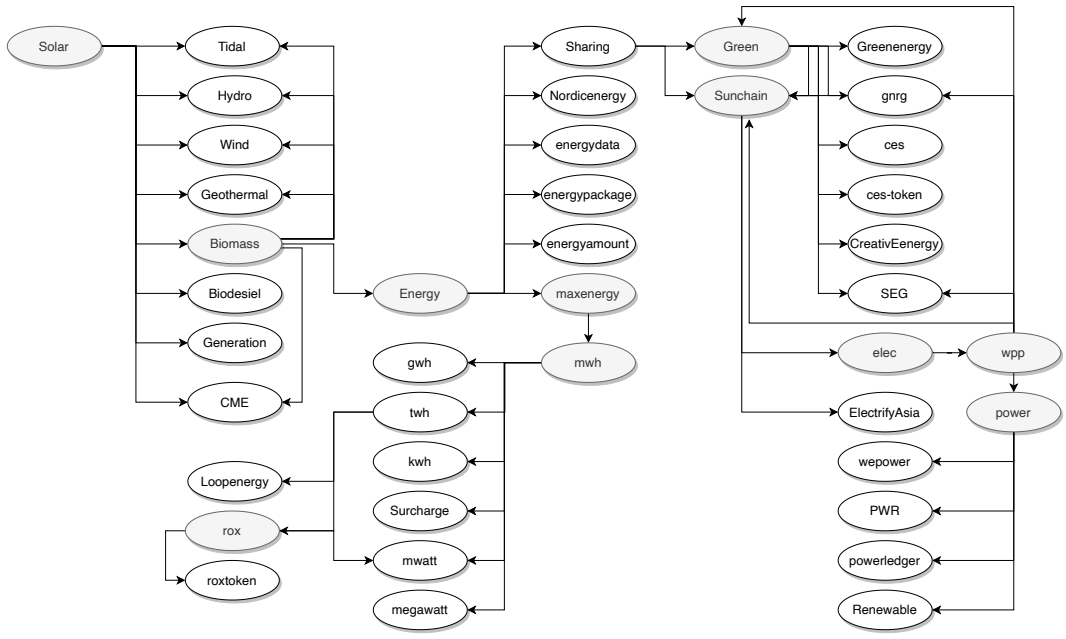


Figure 2.3: Graph of domain-specific terms.

Table 2.1: Word2Vec Embeddings.

| **Term** | **Matches** | **Similarity** |
|---|---|---|
| Solar | Tidal | 0.8525 |
| | Hydro | 0.8455 |
| | Wind | 0.8441 |
| | Geothermal | 0.8300 |
| | Biomass | 0.8185 |
| | CME | 0.8181 |
| | Biodiesel | 0.7907 |
| Biomass | CME | 0.9951 |
| | Geothermal | 0.9932 |
| | Biodiesel | 0.9901 |
| | Hydro | 0.9817 |
| | Tidal | 0.9701 |
| | Wind | 0.9295 |
| | Energy | 0.8791 |
| Sunchain [128] | GNRG [43] | 0.9526 |
| | CES [14] | 0.9034 |
| | ELEC [30] | 0.8951 |
| | ElectrifyAsia [30] | 0.8840 |
| | WPP [159] | 0.8815 |
| | CreativeEnergy [22] | 0.8789 |
| | SEG [20] | 0.8620 |

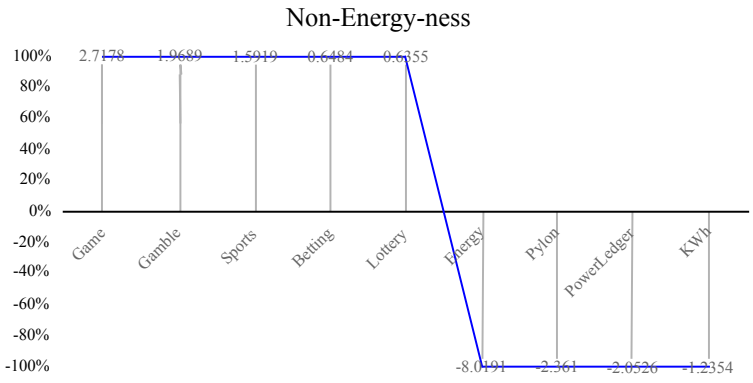


Figure 2.4: Energy context assessment by logistic regression.

the highest level of granularity in the classification. Moreover, pre-processing of raw unstructured text is required for content analysis.

The pre-processing of the source code includes cleaning, normalizing, and stemming to remove stop words and semantic-irrelevant terms and to break down the words to their roots for detection of semantic similarities in feature extraction [143].

### 2.5.2 Building a Domain Corpus

In NLP, the frequency of terms is important to analyze the context. As a result, most feature extraction methods need a corpus of annotated terms paired with an ensemble of algorithms to learn the significance of a term in a document [91]. The process of corpus development and energy token extraction is discussed in this section.

Machine learning has been widely adopted for the semantic and syntactic analysis of text. As a result, most machine learning approaches require a corpus of annotated text for the underlying algorithm to learn the significance of each term in the document [41]. The corpus must be a balanced, representative collection of terms about a specific topic [4]. The vocabulary of size 376 and energy corpus was created using domain knowledge and by extracting domain features and terms from both the comment and code segments of the contracts. Text analysis and feature extraction methods include different levels of granularity including document level, sentence level, and word level

analyses.

In the course of developing the energy corpus and processing the smart contracts, we analyzed source codes at the lexical level. From a lexical standpoint, featuring a domain-specific corpus facilitates capturing the terms and statements that correspond to a specific application domain with prototypical measures at syntactic and semantic levels. Moreover, a domain-specific term distinguishes itself by the relative degree of closure of its lexicon, which signifies that, unlike a general corpus, a domain-specific vocabulary is nearly finite. Since certain syntactic structures and classes are more prevalent in a given application domain than in general corpus, we aim to create key energy terms, tokens, and attribute tags that will facilitate interpreting the context of the code and retrieving energy contracts. Moreover, semantics serve a limited role in smart contract development and using keywords allows a surface-level interpretation of source codes.

Using term frequency and relevance, the corpus is intended to connect a knowledge base as a dictionary to the source code text. The corpus will be updated progressively as new energy contracts are identified during the process.

### Extracting Energy Tokens

Tokens are the most prominent parts of Ethereum smart contracts. On the blockchain, cryptocurrencies serve a comparable function to cash, whereas cryptographic tokens serve as universal tools for managing digital assets. ERC-20 is one of the most widely used Ethereum tokens, adopted by a multitude of digital currencies [146].

The advent of the standardized ERC-20 contributed to the emergence of Ethereum-based utility tokens and a tokenized economy [41]. Accordingly, energy tokens are widely adopted across smart contracts representing tokenizable values for peer-to-peer energy transactions and keeping the values fungible across energy application networks [40, 138]. These tokens are minted and distributed amongst stakeholders with the physical capacity to contribute services to the energy network. For instance, Power

Ledger (a blockchain-enabled energy trading platform) employs POWR tokens to transact energy and trade environmental commodities [103]. As a result, incorporating energy tokens in energy corpus will facilitate the identification of energy contracts.

Following the initial filtering using energy tokens, feature extraction is performed to identify additional properties of energy contracts through text analysis methods discussed in the following section.

### 2.5.3 Embedding Layer

The main challenge in active learning is selecting the most insightful data instances to label and use to begin training. The choice of embeddings requires special consideration since we are parsing source code.

As mentioned earlier, source code deviates from a genuine text in that it features distinct granularity levels and lacks high-quality contextual information at the document level. NLP and word embeddings have recently seen considerable advances. However, source code processing requires semantics knowledge at the concept level rather than unique occurrences in text. At the same time, instance-specific embeddings (as introduced in BERT and similar approaches) are best suited for language translation and search engine queries [134]. Hence, feature selection techniques are useful to identify and eliminate unnecessary and irrelevant subsets of features [73]. Word2Vec [61] and Term Frequency–Inverse Document Frequency (TF-IDF) [105] were used in this work as feature extraction methods. They analyzed over 10,000 smart contracts to facilitate preliminary filtering. Word2vec is a pre-trained word embedding neural network, effective in text classification with small corpus, as in our case. The Word2Vec embedding layer keeps the semantic and syntactic information of codes and comments and it predicts the context of the terms. Using Continuous Bag of Words (CBOW) as the underlying architecture, the semantic correlation of the existing terms in the corpus was evaluated to find the closest match [167]. CBOW quantifies the frequency of the terms in the document by assigning each term a value

representing the occurrence of that feature.

The corpus is updated by terms with the highest semantic similarity scores, as illustrated in Table 2.1. Using a lexicon of energy-relevant terms, the occurrence of each term and their corresponding semantic correlation score are factored as measures of the energy-ness of the contract. Subsequently, the pre-processed contracts are passed through an embedding layer stacked in front of the classification model. TF-IDF is used as a sparse embedding layer to extract features from the labeled data. It works by penalizing frequently occurring terms in the source code to identify prominent yet infrequently used terms that prevail over the context. Using TF-IDF, each term in the source code is assigned a weight that determines the significance of the term in the source code based on its frequency and inverse document frequency used for training the classifier.

### 2.5.4 Baseline Models

Determining the best classifier is an imperative yet challenging decision in any text classification workflow. It needs to take into account many aspects, including the data composition, scalability of training, and run-time efficiency. In this study, NB, LR, and SVM are used as baseline models for classification of the Ethereum smart contracts [55].

#### Logistic Regression

LR is a discriminative, probabilistic classifier that is commonly employed in NLP as a supervised machine learning model. Based on its core assumption that dependent and independent variables do not have a linear relationship, LR examines the relationship between categorical variables using a logistic function. It requires a training corpus to detect discriminating features between the desired classes.

The input corpus is used by LR to learn the domain's verbal intuition and syntactic literature, as well as to retrieve document features and biased terms. Each input feature

is assigned a weight that represents its importance in the classification decision. LR assigns higher weights to the primitive terms, although it is not capable of generating an instance of these terms on its own. The bias term, commonly referred to as the intercept, is also added to the weighted inputs. This implies that energy terms are negatively associated with the non-energy decision, as illustrated in Figure 2.4.

Instead of determining similarity, LR takes into account the distance between the energy and the non-energy contracts. Following that, gradient descent is used to iteratively update the weights in order to minimize the cross-entropy loss which is a convex optimization problem. Hence, the algorithm's resistance to correlated characteristics contributes to a higher classification precision.

**Naive Bayes**

Another probabilistic, supervised classifier employed in this study is NB. It determines the likelihood of a label based on previously observed characteristics and their conditional independence. Using the Bayes theorem, this model identifies the correlation between conditional probabilities and statistical quantities. As an incremental approach, NB is predicated on a theory that all attributes are independent and any context disregarding this conditional independence principle deteriorates its performance.

The source code is transformed into a feature vector as an input for naive Bayes to be trained on the training set, estimating the likelihood of energy-ness given each feature. Generally, features can be developed by analyzing the training set while keeping linguistic intuitions and the domain-specific linguistic literature in mind. Developing complex features that are variations of a number of primitive features is especially useful, as illustrated in Figure 2.3.

A thorough assessment of errors on the training set often yields perspectives on these features. NB generates the probability of each feature for each class, such that the probability of each feature can be optimized to project energy-ness or non-energy-ness

of the smart contracts.

**Support Vector Machine**

The last supervised learning approach used in this research is SVM. It has proven to be an effective method for pattern recognition and text classification. As a discriminant classifier with a statistical learning paradigm, SVM attempts to capture the optimum trade-off between complexity and learning to ensure maximum generalization and minimum structural risk.

The source code is perceived as a bag of words and each term is associated with a feature where the significance of the feature is determined by the frequency by which it appears inside the contract using TF-IDF. Once feature vectors are obtained,

SVM transforms the training set into a multidimensional space to create a hyper-plane. The optimal position of the hyper-plane is directly affected by the data points closest to the decision boundary as they are the most challenging to identify. In addition, a subset of the training set is used as support vectors in the decision function, making it memory efficient. Using a higher dimension, SVM distinguishes the classes with the highest marginal distance, establishing a decision boundary to optimize the classification accuracy.

## 2.6 Evaluation Results

Tables 2.2 and 2.3 demonstrate how each model performs on energy and non-energy contracts independently, as well as the overall accuracy of each algorithm using precision, recall, and the F1-score [45]. The obtained results show that LR produced the maximum accuracy of 98.34%, 97.53% precision, 98.78% recall, and 98.12% F1-score. Figure 2.4 depicts the LRs effective energy assessment of the frequently encountered terms in the source codes.

The model is fed the TF-IDF features and their corresponding coefficients, which translate to a weighted combination of input features. This is to determine the

importance of the feature in the overall logloss calculation. Hence, the probability of the contract being a non-energy contract increases as the logloss increases, while the probability of the contract being an energy contract increases as the logloss decreases. As a result, the term "Energy" has been appointed a $-8.0191$ correlation score with non-energy terms and is regarded as the most energy-related term. Pylon [1], which has been further identified as one of the dominant energy tokens adopted in several energy transactions, has been classified as an energy-related term with a $-2.361$ correlation score with respect to non-energy terms. On the other hand, terms such as game, gamble, sport, betting, and lottery that embody dominant application areas for smart contracts are classified as non-energy terms [57].

LR achieved substantial results since the classification task is fundamentally a binary problem. In addition, the probabilistic structure of LR allows the use of the likelihood ratio for reducing the costs associated with misclassification. The primary difference between LR and NB as the best performing algorithms is that LR is a discriminative classifier and NB is a generative classifier [114]. As a result, the success of LR over NB can be traced back to LRs robustness to correlated features and NBs intense conditional independence theorem. Although NB is acknowledged for its fast convergence, it demonstrated relatively high errors compared to LR. Nevertheless, because of its comparable outcomes and effortless training, NB remains a viable technique for use on small datasets [116].

SVM produced satisfactory results in the contract classification, with 87.23% accuracy. SVM is able to generalize because, as a probabilistic method, it does not penalize instances in which the correct decision is made with a reasonable degree of certainty. However, SVM underperformed in comparison to the other algorithms since it aims to maximize the perpendicular space between the two edges of the hyperplane to reduce the risk of generalization errors. As a result of the high correlation between smart contracts, the marginal distance of the data points decreased, resulting in a

Table 2.2: Class-specific metrics from the confusion matrix report.

(a) Energy Class

| Model | Precision | Recall | F1-Score |
|---|---|---|---|
| LR | 0.95 | 1.0 | 0.97 |
| Naive Bayes | 0.86 | 0.92 | 0.89 |
| SVM | 0.97 | 0.77 | 0.86 |

(b) Non-energy Class

| Model | Precision | Recall | F1-Score |
|---|---|---|---|
| LR | 1.0 | 0.98 | 0.99 |
| Naive Bayes | 0.91 | 0.86 | 0.88 |
| SVM | 0.81 | 0.98 | 0.88 |

higher generalization error and a lower accuracy.

Subsequently, a fraction of the identified energy contracts were analyzed to capture any patterns in code segment distribution, prevalent adoption of specific functions, and recurring contracts across the Ethereum network [102]. Figure 2.5 depicts the most common energy tokens used to facilitate transactions in energy smart contracts. POWR is a utility token that grants access to the Power Ledger platform and its peer-to-peer features and serves as the ecosystem's fuel [103]. Similarly, WPP Energy is a publicly available blockchain-based renewable energy investment platform that offers peer-to-peer smart-contract-enabled transactions using WPP tokens in an effort to promote the use of cryptocurrency in the energy market [159]. WePower is another blockchain-based green energy trading platform that enables energy suppliers to gain capital for green energy efficiency through smart contracts and WPR tokens. These tokens are indicative of the energy produced by producers in the days to come, giving buyers the opportunity to invest in renewable energy [95].

Table 2.3: Performance of the baseline algorithms on a full-feature model.

| **Model** | **Accuracy** | **Precision** | **Recall** | **F1-Score** |
|---|---|---|---|---|
| LR | 0.9834 | 0.9753 | 0.9878 | 0.9812 |
| Naive Bayes | 0.8910 | 0.8923 | 0.8910 | 0.8909 |
| SVM | 0.8723 | 0.8900 | 0.8723 | 0.8709 |

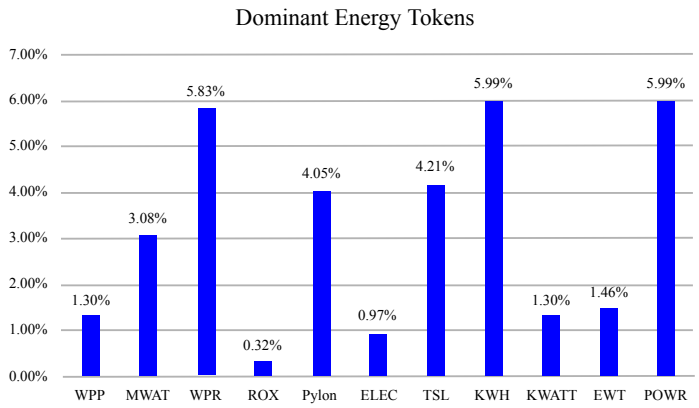


Figure 2.5: Dominant energy tokens.

Finally, potential discrepancies between the code segments of both categories were examined, selecting 20 energy smart contracts and 20 non-energy smart contracts. Figure 2.6 illustrates the comparative results obtained from the distribution of contract code segments. When compared to non-energy smart contracts, the results obtained on the distribution of energy code segments imply that the development of energy contracts tends to prioritize the adoption of contracts and libraries over interfaces.

The number of Logical Lines of Code (LLOC) (excluding comments and empty lines), the number of Source Lines of Code (SLOC), the Number of Functions (NF), the deepest Nesting Level (NL), the number of parameters (PAR), and the number of Comment Lines of Code (CLOC) results also validate the prevalent use of identical contracts with minor adjustments and demonstrate how the lines of comments are heterogeneous and not necessarily proportionate to the length of the code. The results further confirm the prominent adoption of StandardToken and ERC20 in both classes upon analyzing dominant contract names and function names across both energy and non-energy smart contracts.

As illustrated in Figure 2.7 and 2.8, the primary differences between the two classes in terms of function names can be attributed to the Ownable, Safemath, Pausable, and Mintable contracts. This indicates that Mintable contracts are not commonly used in energy contract development, since Mintable tokens feature a non-fixed total supply, allowing the token issuer to mint additional tokens. On the other hand, Ownable, Safemath, and Pausable are not identified as dominant non-energy contracts. Owanble contracts may be utilized for lowering gas costs and binding configuration functions to specific external addresses and are widely adopted in the energy sector.

The SafeMath library examines whether an arithmetic operation will result in an integer overflow/underflow. Energy contracts use Safemath to send an exception and rollback the transaction. Pausable contracts are another common practice among energy contracts, allowing the owner of a Pausable contract to halt and restart functions. The owner can pause the functionality at any time; thus, users may be

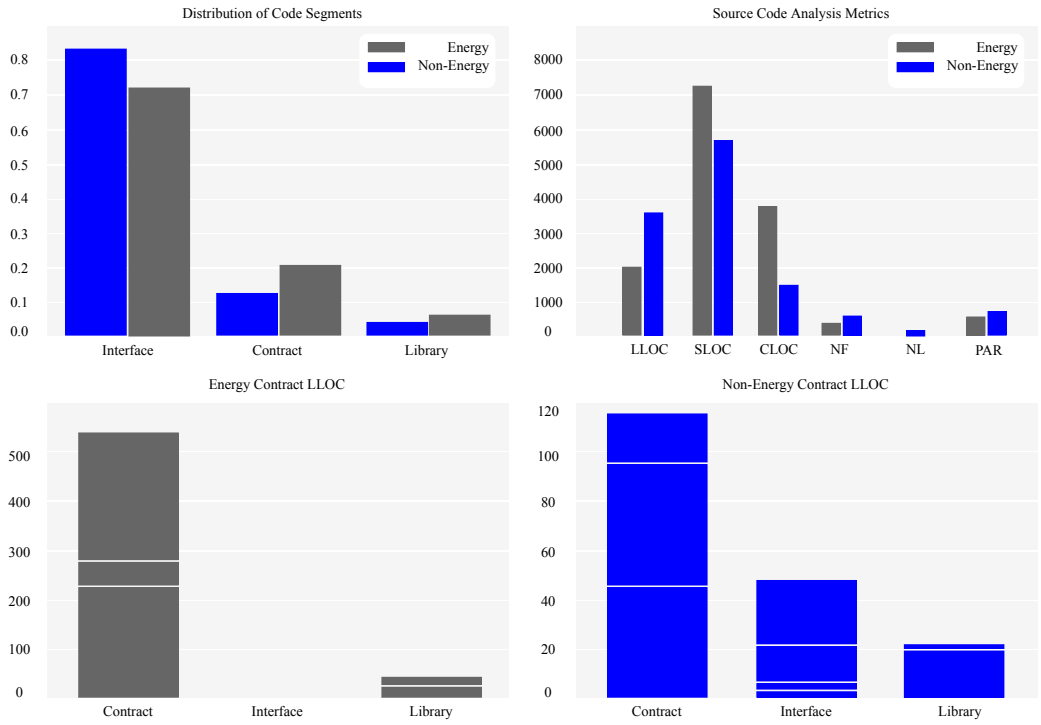


Figure 2.6: Code segment analysis I

hesitant to utilize the corresponding dApp, making this a drawback in design for energy contracts. Although from a vulnerability analysis standpoint, the lack of a pause mechanism requires vulnerable contracts to be aborted while an alternate instance becomes available on the blockchain. Finally, the same analysis among functions in both classes entails the adoption of comparable functions across each category, with more prevalent adoption among energy contracts, as illustrated in Figure 2.7.

## 2.7 Conclusions

Blockchain technology has brought innovation to a wide array of industries. The number of transactions on the Ethereum blockchain is approaching half a billion, turning Ethereum into the largest smart contract blockchain platform. Unlike traditional contracts, smart contracts are not written in a natural language, making it difficult to determine their content. As a result, smart contract classification based on the application domain and transaction context provides greater insight into the syntactic

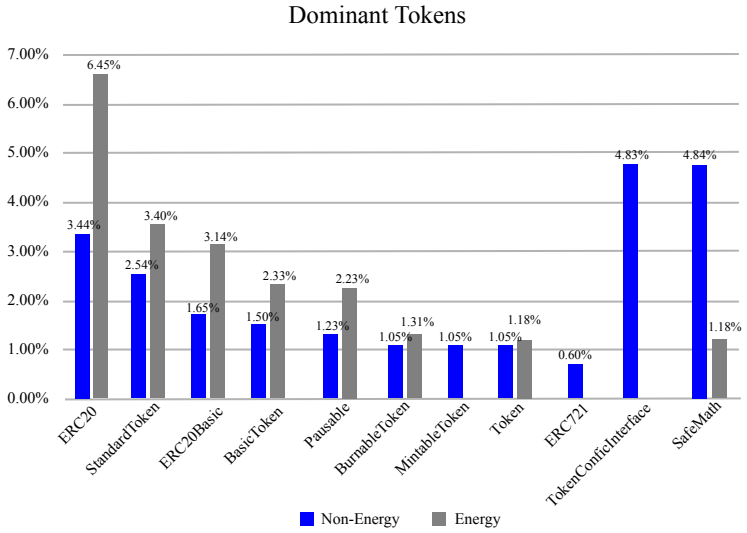


Figure 2.7: Code segment analysis II

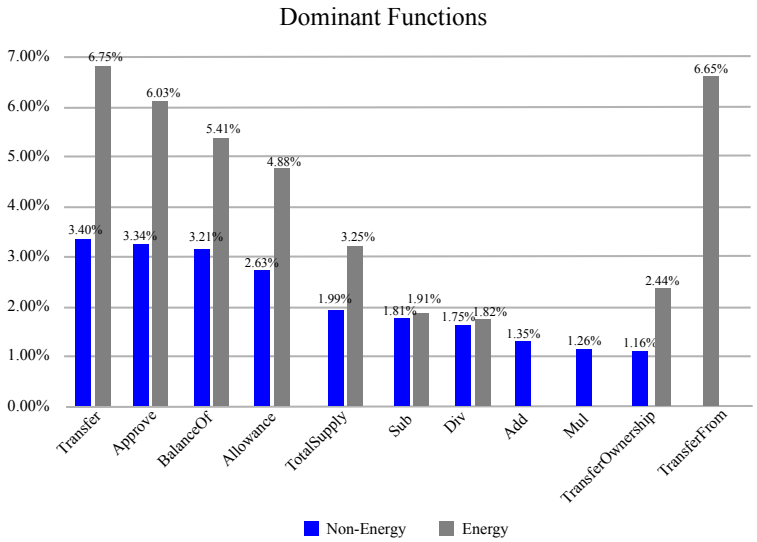


Figure 2.8: Code segment analysis III

and semantic properties of that domain. With the progression towards a more decentralized and dynamic energy system, the impact of blockchain-enabled smart contracts in transactive energy systems has gained prominence. As a result, it is imperative to analyze the energy smart contract feature space to gain more insights into the characteristics of contracts deployed for energy transactions.

Analyzing over 10,000 smart contract solidity source codes, this study proposes an approach to discriminate energy smart contracts using the publicly accessible Ethereum source codes. NLP and machine learning classification algorithms are employed to detect and properly label energy smart contracts. To begin, a domain-specific embedding layer is generated to identify and analyze energy tokens and energy-related terms. Subsequently, both the energy corpus and categorical attributes are employed as baselines for the training of the classification algorithms. Logistic regression, naive Bayes, and Support Vector Machine are implemented as classifiers. The classification performance of each algorithm is then evaluated using accuracy, precision, recall, and F1-score metrics. Energy smart contracts are detected with up to 98.34% accuracy, with LR outperforming the other algorithms. Detected contracts are further examined to discern any discrepancies or patterns in the distribution of code segments, the predominant use of specific functions, and recurring contracts across the Ethereum network.

We anticipate that the proposed approach will help to establish the groundwork for innovative solutions for domain-specific classification and vulnerability detection of smart contracts. Looking exclusively into the grammatical, symbolic, and arithmetic characteristics of energy smart contracts may facilitate the identification of vulnerability features that may have gone undetected in previous studies. Subsequently, machine learning models can be employed for vulnerability assessments of the energy contracts at the function level. To improve the accuracy of the existing vulnerability detection models, the implications of integrating the conventional pattern extraction methods with machine learning models can also be investigated.

# Chapter 3

# Evaluation of Smart Contract Vulnerability Analysis Tools: A Domain-Specific Perspective

## 3.1 Abstract

With the widespread adoption of blockchain platforms across various decentralized applications, the smart contract's vulnerabilities are continuously growing and evolving. Consequently, a failure to optimize conventional vulnerability analysis methods results in unforeseen effects caused by overlooked classes of vulnerabilities. Current methods have difficulty dealing with multifaceted intrusions, which calls for more robust approaches. Therefore, overdependence on environment-defined parameters in the contract execution logic binds the contract to the manipulation of such parameters and is perceived as a security vulnerability. Several vulnerability analysis tools have been identified as insufficient to effectively identify certain types of vulnerability. In this chapter, we perform a domain-specific evaluation of state-of-the-art vulnerability detection tools on smart contracts.

A domain can be defined as a particular area of knowledge, expertise, or industry. We use a perspective specific to the area of energy contracts to draw logical and language-dependent features to advance the structural and procedural comprehension of these contracts. The goal is to reach a greater degree of abstraction and navigate

the complexities of decentralized applications by determining their domains.

In particular, we analyze code embedding of energy smart contracts and characterize their vulnerabilities in transactive energy systems. We conclude that energy contracts can be affected by a relatively large number of defects. It also appears that the detection accuracy of the tools varies depending on the domain. This suggests that security flaws may be domain-specific. As a result, in some domains, many vulnerabilities can be overlooked by existing analytical tools. Additionally, the overall impact of a specific vulnerability can differ significantly between domains, making its mitigation a priority subject to business logic. As a result, more effort should be directed towards the reliable and accurate detection of existing and new types of vulnerability from a domain-specific point of view.

## 3.2 Introduction

In essence, smart contracts are self-executing collections of codified agreements deployed on decentralized networks of blockchain. Smart contracts support decentralization through automated execution, blockchain integration, and immutable protocols. They can also contribute to trust through transparency, automated execution, immutable history, and cryptography.

As event-driven programs, smart contracts facilitate trusted transactions, allowing anonymous parties to exchange digital assets or data [160]. The smart contract is the main pillar of Ethereum and is widely adopted in various business domains. As a proclaimed framework for integrating the execution of smart contracts, Etheruem offers great capacity in the development of decentralized applications (DApps). Most Ethereum smart contracts are written in Solidity, a high-level object-oriented programming language, and then compiled into bytecode for execution using the Ethereum Virtual Machine (EVM). The EVM creates a level of abstraction between the executing code and the machine on which it runs, isolating the DApps and their corresponding hosts [120].

Solidity smart contracts, like any other program, are susceptible to bugs, vulnerabilities, and security flaws caused by a lack of security patches [72]. However, smart contracts are associated with immutability features that preclude any modification after deployment, thereby enforcing the "code is law" principle [78]. Consequently, it is necessary to verify that a smart contract does not contain hidden programming defects that could have severe security implications. These defects could be maliciously exploited by an attacker to initiate unintended processes the smart contract was not set up to perform.

As more blockchain-based services are created, there is greater emphasis on the reliability of smart contracts. In an analysis of approximately one million Ethereum smart contracts, 34,200 have been identified as vulnerable [98]. Therefore, the early detection of vulnerabilities prior to the deployment of smart contracts is extremely important. Therefore, automated methods and tools have been developed to analyze smart contracts to detect vulnerabilities and bad coding practices [3, 111].

The performance of comparable vulnerability analysis tools varies substantially among different smart contract analysis studies. This is explicable through different experimental settings and performance metrics. Consequently, evaluating the detection competencies of these tools is very difficult.

Our research has shown that existing research on the effectiveness of benchmark vulnerability analysis tools has not considered the application domain or the purpose of the contracts under assessment. Although correlations between smart contract categories and underlying vulnerabilities were identified in a recent study [60], security defects in energy contracts were not factored into the evaluation.

To fill this gap, the primary focus of this study is the analysis of smart contracts deployed in transactive energy systems. The proposed vulnerability analysis workflow allows the investigating of the associations between various contract types and corresponding defects. Consequently, a domain-specific evaluation of both static and dynamic analysis tools is carried out using curated contracts as the benchmark. We

have used state-of-the-art vulnerability analysis tools on energy and non-energy smart contracts to answer the following questions:

- Which tool performs best in analyzing the vulnerability of smart energy contracts?
- Do energy contracts contain more vulnerabilities and poor coding practices compared to other classes of contracts?
- Are there domain-specific security flaws that existing tools fail to detect?
- Are certain state-of-the-art vulnerability analysis tools more effective in specific application domains?
- Is there any benefit to developing domain-specific vulnerability detection tools?

## 3.3 Background and Motivation

Given the global interest in blockchain, researchers are investigating the vulnerabilities of smart contracts as they form the cornerstone of blockchain development. Zeus [65] evaluated over 22,400 solidity smart contracts and found that approximately 94.6% of them had security defects. Another database containing blockchain security defects constructed by researchers denoting smart-contract-related instances accounts for 22% of all incidents [18, 26]. Smart contracts have complex time and order dependencies. Hence, inconsistencies in the logic of a contract code contribute to vulnerabilities and the incorrect execution of the smart contract [72]. As a result, blockchain-oriented software engineering is necessary to avoid defects and ensure effective programming practices.

Smart contract defects encompass a combination of security-related problems as well as design deficiencies that could impede the implementation or increase the possibility of future vulnerabilities or failures. Defects often result in the smart contract producing an erroneous or undesirable outcome or causing it to operate in directions that were

not intended. Hence, the identification and elimination of these defects improve the software reliability and development process [15].

The severity of defects can be categorized into three distinct types based on their implications and the likelihood of causing a financial loss. Direct monetary losses are classified as higher-grade defects, whereas risks of monetary losses that arise from unauthorized or unintended contract operation are rated as medium-severity defects. Defects that cause only superficial problems (such as poor accessibility or resource depletion), do not interfere with regular operations, and do not result in financial losses, are classified as low-severity defects. However, the broad spectrum of defects makes it challenging to define them precisely. Hence, defect patterns are incorporated as a conceptual representation to depict defect characteristics and attributes [66].

A pattern is an abstraction from a concrete design that recurs in predetermined, nonarbitrary instances. Patterns encompass an overarching outline of a persistent issue and an applicable solution with explicit goals and limitations. Different assessment expressions for smart contracts can be implemented by pattern design. Each defect type could entail a distinct collection of code segments and impose a varying set of complications. Consequently, the development of defect patterns is crucial for accurately and effectively conveying these security defects. Code Elements (CE), Relationship Restrictions (RR), and Element Retrictions (ER) make up the majority of the defect pattern. The pattern promotes the evaluations required to improve the reusability of the contract code. Consequently, the validation and updating process for smart contracts can be simplified by reusing the corresponding verification rules [44].

However, as smart contract structures becomes increasingly complex, a multitude of vulnerabilities are emerging and expert-defined rules cannot keep up with constant vulnerability updates. The resulting overlay of expert-defined rules leads to substantial false alarm rates. This makes the rule-based detection of general vulnerabilities impractical.

Several predeployment smart contract security analysis frameworks have been

developed in response to the substantial economic implications of defects in smart contracts [28]. However, the primary concern remains the efficient and timely detection of smart contract vulnerabilities. As the functionality of smart contracts expands, their new attributes contribute to new types of security flaws. These flaws, in turn, allow for complex attacks such as the new ERC777 reentrancy and cross-chain attacks [2].

Integration and acceptance tests developed for defect detection require extensive technical knowledge of the blockchain framework. Any reliance on environment-defined parameters in the execution logic of the contract binds the contract to the manipulation of these parameters and is considered a security flaw [8]. Addressing prevalent smart contract vulnerabilities requires a syntactic and semantic understanding about the compromised contract.

The literature suggests that smart contract classification with respect to the application domain and transaction context provides additional information on the syntactic and semantic properties of the domain [118].

To identify new vulnerabilities, researchers must continue to expand their detection prospects for the methods encoded in smart contracts that contain their business logic. This study examines the benefits of incorporating business logic and domain knowledge in the design and development of vulnerability analysis tools. We anticipate that its results will help to establish the groundwork for innovative solutions for the domain-specific classification and vulnerability analysis of smart contracts.

## 3.4 Vulnerability Analysis Tools

The smart contract vulnerability analysis tools can be classified into two main categories, static and dynamic [71]. Static analysis is the process of analyzing a program without executing it. This method can be applied to both source code and bytecode representations of smart contracts.

Dynamic analysis, on the other hand, is a run-time environment method that screens the contract's behavior during execution for potential vulnerabilities or security

breaches. Fuzzing is a form of dynamic analysis that passes defective data to the executed smart contract to examine its response to the malformed input. Using the Smartbugs [121] interface, this study targets a series of static and dynamic analysis tools including Mythril, Slither, Smartcheck, Honeybadger, Osiris, Solhint, Oyente and Conkas, and Confuzzius.

- **Honeybadger [121]** is an Oyente-based honeypot detection system that relies on symbolic execution and well-defined heuristics.

- **Osiris [141]** is another Oyente-based tool that claims to be capable of finding previously unknown critical vulnerabilities in some cases. Using symbolic execution coupled with taint analysis, Orisis offers the detection of a diverse range of defects with improved detection specificity.

- **Solhint [123]** was proposed as a linting tool for solidity smart contracts. Using pre-configured patterns and rulesets, it offers a good coverage of known security defects.

- **Smartcheck [137]** validates contracts against XPAth queries using their XML representation. This intermediate representation facilitates the localization of detections across the source code to provide complete code coverage.

- **Oyente [86]** leverages operational semantics to search for execution traces in the code where the transaction sequence has affected the Ether flow or the result of computations is dependent on timestamps.

- **Conkas [121]** is another static analysis method that incorporates control flow graphs (CFGs) as intermediate representations for symbolic execution. If the user does not specify the dependency files, Conkas is not capable of tracing the vulnerabilities encapsulated in the library files.

- **Mythril [21]** is intended to uncover common security issues and cannot detect the concerns ingrained in business logic. It incorporates concolic, taint, and control flow analysis to search for attributes that cause vulnerabilities in smart contracts.

- **Slither [36]** employs its own internal representation language for an intermediate representation and performs data flow and taint analysis for information retrieval and refinement. Slither determines a set of predefined analyses and a Static Single Assessment (SSA) in a multistage procedure. With the Abstract Syntax Tree (AST) as input, it can provide enhanced information to the other components and simplify the computation of a diverse array of code analysis.

- **Confuzzius [140]** is the first hybrid fuzzer that integrates evolutionary fuzzing with constraint solving to explore both shallow and deep fragments of contracts. Using dynamic data dependency analysis, Confuzzius can derive transaction sequences that lead to states with implicit security flaws.

## 3.5 Domain-Specific Perspective

Several attempts have been made to investigate smart contact vulnerabilities; however, no research has examined the correlation between different types of contract and their corresponding security flaws. A recent study reported a positive association between smart contract categories and their vulnerabilities.

Giacomo et al. [60] suggest that gambling contracts are consistently associated with bad randomness. However, the smart contract classification in this study fails to consider many other application domains, including transactive energy systems.

Smart contracts have a broad range of applications in the energy industry, including monitoring the production, distribution, and consumption of energy. These codified agreements can also be used for the monitoring of carbon credits and the administration of peer-to-peer energy markets. These markets represent an architectural transition

from traditional centralized energy distribution models. They are well positioned to transform the energy landscape, with smart contracts enabling secure and transparent transactions between energy suppliers and end users.

The inherent trust and transparency of blockchain technology, combined with the automated characteristics of smart contracts, fosters a decentralized energy exchange ecosystem that allows individuals to participate in direct energy trading, consumption, and reimbursement. In a comparable study, smart contracts used in transactive energy systems are examined to derive the business logic and disclose the features of the contracts and the verbal intuition of the energy contracts [76].

Energy smart contracts encapsulate the terms and conditions governing a contract among counterparties to regulate the transactions of electricity such as processing market bids, billing, and optimal pricing for double-auction.

The code snippet illustrated in Listing 1 is an energy contract instance called *EnergyTrading* that authorizes energy transactions by establishing the price and availability using *setEnergyPrice* and *setEnergyAmount* functions. This contract offers another function called *SellEnergy* that allows for keeping track of the energy sold and ensures that the balance does not exceed the available energy. *EnergySold* is another event used in this contract to monitor sales.

Listing 3.1: Sample energy smart contract.

```
pragma solidity ^0.8.0;

contract EnergyTrading {

 uint256 public energyPrice;
    uint256 public energyAmount;

    event EnergySold(address seller, uint256 amount, uint256 price);

    constructor(uint256 initialPrice) public {
        energyPrice = initialPrice;
    }

    function setEnergyPrice(uint256 newPrice) public {
        energyPrice = newPrice;
    }

```

```
18     function setEnergyAmount(uint256 newAmount) public {
19         energyAmount = newAmount;
20     }
21
22     function sellEnergy(uint256 amount) public {
23         require(amount <= energyAmount, "Not enough energy available
               for sale");
24         energyAmount -= amount;
25         msg.sender.transfer(amount * energyPrice);
26         emit EnergySold(msg.sender, amount, energyPrice);
27     }
28 }
```

The DOT-formatted control flow graph of the *EnergyTrading* contract is depicted in Figure 3.1 using Surya [21]. This is a common method intended to perform manual contract analysis, inspecting the contract's complexity using control flow graphs and inheritance graphs. This approach to contract development minimizes complexity while avoiding security breaches.

As shown in Figure 3.1, the contract functions are not interconnected and cannot be called by external contracts. However, research shows that this is not a recurring pattern for energy contracts. This is because these contracts feature various levels of inheritance that embed complex computations. Additionally, energy contracts encompass a plethora of external and internal calls that are related to the complexity of the contract [69]. Consequently, smart contract vulnerability detection research would benefit from an analysis of the effectiveness of vulnerability detection methods in connection with certain vulnerabilities and application domains.

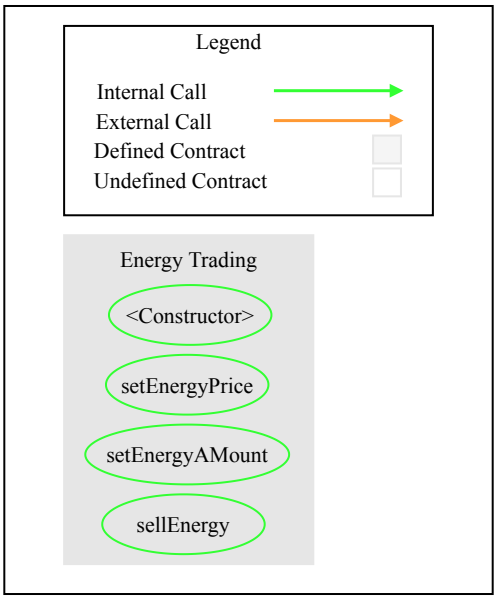


Figure 3.1: Control flow graph.

As a result of different experimental setup and performance criteria, the performance of benchmark tools varies between different vulnerability analysis studies. Tailoring the experimental setup to a target domain enables the search for critical security breaches in contracts with comparable violation structures and behavioural patterns.

## 3.6 Analysis

As shown in Figure 3.2, the analysis in this study is carried out using three distinct data sets, each containing 20 smart contracts. The first data set, used to benchmark smart contracts, is the curated data set of SB [121]. The primary objective of this data set is to provide a collection of previously identified vulnerabilities that can be used to evaluate the performance of analysis tools.

The appointed contracts encompass vulnerabilities such as reentrancy, access control, time manipulation, bad randomness, and arithmetic issues. The other two data sets encompass 20 manually labeled energy and 20 non-energy contracts from Etherscan. As stated earlier, energy contracts are used to automate energy trading activities, while non-energy contracts are formulated to regulate transactions in other application

domains, such as gaming, gambling, finance, and so on. The vulnerability analysis is carried out on both energy and non-energy categories using all analysis tools. The analysis can be broken down into two segments: benchmark and energy, each of which is discussed in the sections that follow.

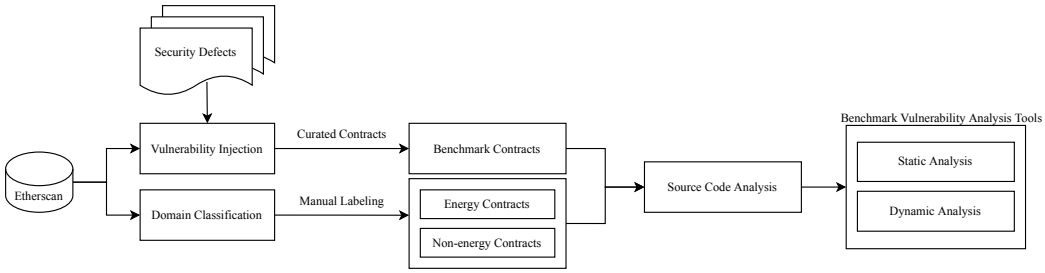


Figure 3.2: Vulnerability analysis workflow.

### 3.6.1 Benchmark

The first part of the experiment serves as a benchmark to evaluate cutting-edge smart contract analysis tools on most critical vulnerabilities using the curated data set. Accordingly, the detection rate of each tool can be evaluated using true positive, false negative, average run-time, and accuracy metrics.

As illustrated in Table 3.1, across all the tools, Slither identifies most defects and has the highest accuracy and the shortest runtime. It is widely regarded as a highly effective tool that incorporates code and constraint-solving mechanisms. Therefore, the obtained results are consistent with previous research on the evaluation of smart contract analysis tools [169]. Honeybadger, with symbolic execution and constraint solving, is the least accurate method, followed by Solhint with code instrumentation. Mythrill is the second slowest method with mediocre accuracy, showing the incompetence of symbolic execution paired with constraint solving [71].

Table 3.1: Analysis results on curated dataset.

| | | Analysis Tools | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Vulnerabilities** | **Metrics** | **Confuzzius** | **Conkas** | **Honeybadger** | **Mythril** | **Osiris** | **Oyente** | **Slither** | **Smartcheck** | **Solhint** |
| **Bad randomness # 1** | Duration | 8.73 | 43.17 | 42.62 | 117.75 | 35.85 | 12.72 | 2.46 | 7.22 | 2.34 |
| | Detection | ✓ | × | × | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Bad randomness #2** | Duration | 5.14 | 4.02 | 77.66 | 35.02 | 4.40 | 9.35 | 2.50 | 8.29 | 2.40 |
| | Detection | ✓ | ✓ | × | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Bad randomness #3** | Duration | 5.88 | 5.73 | 4.73 | 23.37 | 5.19 | 4.57 | 4.68 | 12.09 | 4.31 |
| | Detection | × | × | ✓ | ✓ | × | × | ✓ | ✓ | × |
| **Bad randomness #4** | Duration | 13.40 | 25.54 | 270.56 | 665.07 | 52.94 | 31.93 | 2.29 | 7.96 | 2.46 |
| | Detection | ✓ | ✓ | ✓ | × | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Access Control #1** | Duration | 5.20 | 4.01 | 4.67 | 29.09 | 3.29 | 3.31 | 1.74 | 9.40 | 2.07 |
| | Detection | × | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Access Control #2** | Duration | 8.69 | 32.12 | 4.97 | 287.82 | 4.57 | 4.49 | 2.58 | 9.42 | 2.65 |
| | Detection | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Access Control #3** | Duration | 12.31 | 7.73 | 4.43 | 75.80 | 4.46 | 3.81 | 3.22 | 12.97 | 5.03 |
| | Detection | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Access Control #4** | Duration | 9.17 | 12.17 | 6.86 | 187.02 | 3.60 | 5.65 | 4.35 | 8.52 | 2.50 |
| | Detection | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Reentrancy #1** | Duration | 26.17 | 556.36 | 22.80 | 64.55 | 106.86 | 56.45 | 3.13 | 103.80 | 7.27 |
| | Detection | ✓ | ✓ | × | × | ✓ | ✓ | ✓ | × | × |
| **Reentrancy #2** | Duration | 8.12 | 3.57 | 2.98 | 66.90 | 3.46 | 4.16 | 2.38 | 8.54 | 2.34 |
| | Detection | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Reentrancy #3** | Duration | 10.48 | 3.77 | 2.67 | 18.28 | 4.61 | 4.57 | 2.57 | 11.23 | 2.04 |
| | Detection | ✓ | ✓ | × | × | ✓ | ✓ | ✓ | × | × |
| **Reentrancy #4** | Duration | 3.70 | 3.03 | 2.18 | 26.53 | 4.16 | 3.10 | 1.80 | 5.17 | 2.09 |
| | Detection | ✓ | ✓ | × | × | ✓ | ✓ | ✓ | × | × |
| **Time Manipulation #1** | Duration | 4.84 | 3.27 | 5.65 | 21.29 | 3.30 | 2.66 | 1.79 | 11.70 | 1.99 |
| | Detection | ✓ | ✓ | × | ✓ | ✓ | × | ✓ | × | × |
| **Time Manipulation #2** | Duration | 20.45 | 3.69 | 4.36 | 271.60 | 3.69 | 5.64 | 2.26 | 14.23 | 4.23 |
| | Detection | × | ✓ | × | ✓ | × | × | ✓ | × | × |
| **Time Manipulation #3** | Duration | 5.51 | 4.20 | 3.61 | 10.61 | 4.29 | 3.04 | 2.79 | 30.98 | 3.02 |
| | Detection | ✓ | ✓ | × | × | ✓ | ✓ | × | × | × |
| **Time Manipulation #4** | Duration | 4.28 | 5.24 | 6.81 | 47.40 | 3.22 | 9.51 | 2.83 | 115.53 | 10.58 |
| | Detection | ✓ | ✓ | × | × | × | × | × | × | × |

Table 3.1: Analysis results on curated dataset (Cont.)

| | | Analysis Tools | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Vulnerabilities** | **Metrics** | **Confuzzius** | **Conkas** | **Honeybadger** | **Mythril** | **Osiris** | **Oyente** | **Slither** | **Smartcheck** | **Solhint** |
| **Arithmetic #1** | Duration | 45.14 | 168.23 | 134.60 | 1164.38 | 68.61 | 22.46 | 2.33 | 66.02 | 1.90 |
| | Detection | ✓ | ✓ | × | × | ✓ | ✓ | ✓ | ✓ | × |
| **Arithmetic #2** | Duration | 5.89 | 4.19 | 3.17 | 44.70 | 3.40 | 4.47 | 2.40 | 6.77 | 2.30 |
| | Detection | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | × |
| **Arithmetic #3** | Duration | 5.99 | 10.97 | 4.43 | 10.24 | 3.96 | 2.23 | 2.95 | 6.18 | 3.42 |
| | Detection | × | ✓ | × | ✓ | ✓ | ✓ | ✓ | ✓ | × |
| **Arithmetic #4** | Duration | 4.67 | 3.45 | 2.29 | 15.65 | 2.95 | 3.12 | 1.94 | 5.63 | 1.91 |
| | Detection | × | ✓ | × | × | ✓ | ✓ | ✓ | ✓ | × |
| **Overall** | Ave Run-time | 11.09 | 41.26 | 30.50 | 144.12 | 16.69 | 9.18 | 2.73 | 20.65 | 3.38 |
| | TP | 0.67 | 0.83 | 0.33 | 0.54 | 0.79 | 0.71 | 0.92 | 0.58 | 0.33 |
| | FN | 0.33 | 0.17 | 0.67 | 0.46 | 0.25 | 0.29 | 0.08 | 0.42 | 0.67 |
| | Accuracy | 0.67 | 0.83 | 0.33 | 0.54 | 0.79 | 0.70 | 0.92 | 0.58 | 0.33 |

### 3.6.2 Energy

To investigate the performance of the analysis tools in different domains, they are applied to classified contracts. As shown in Figure 3.3, the results suggest that, among a comparable number of contracts in both categories, the energy contracts contain a relatively larger number of defects [34, 112]. Consequently, energy contracts take longer to process, as presented in Figure 3.4.

The demographic profiles of the source code depicted in Figure 3.5 imply that the number of LLOC, the depth of NL, and the Number of Attributes (NA) and statements Number of Statements (NOS) are higher among non-energy contracts. Similarly, Coupling Between Object (CBO) classes appears more frequently among non-energy contracts.

Energy contracts, on the other hand, have a Deeper Inheritance Tree (DIT) and more lines of code (SLOC). Correspondingly, energy contracts are more time-consuming to process, with Mythril being the slowest tool across both categories, which is consistent with the results obtained from the curated data set. Furthermore, Osiris, Honeybadger, and Oyente endured evident instances of execution failures and timeouts in both categories. As demonstrated in Figure 3.6, an analysis of energy contracts

resulted in far more timeouts, with Osiris having the most timeouts, followed by Oyente and Honeybadger.

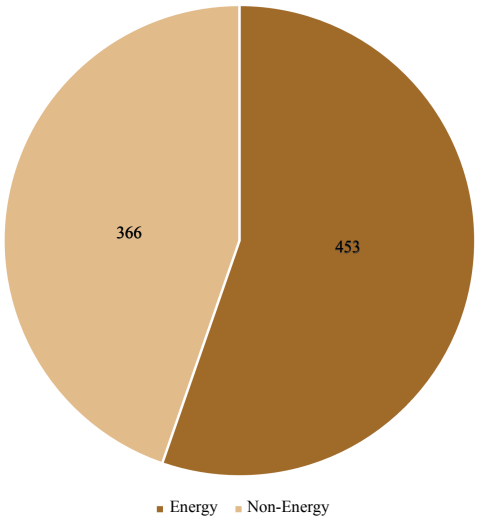


Figure 3.3: Vulnerabilities across different domains.

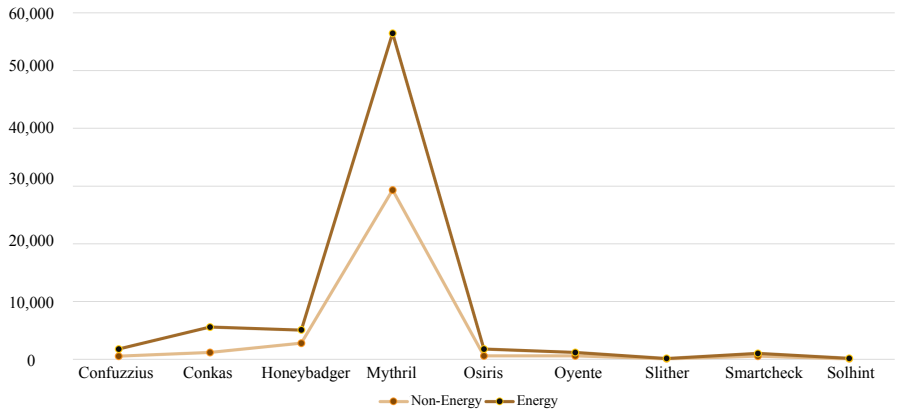


Figure 3.4: Contract processing time.

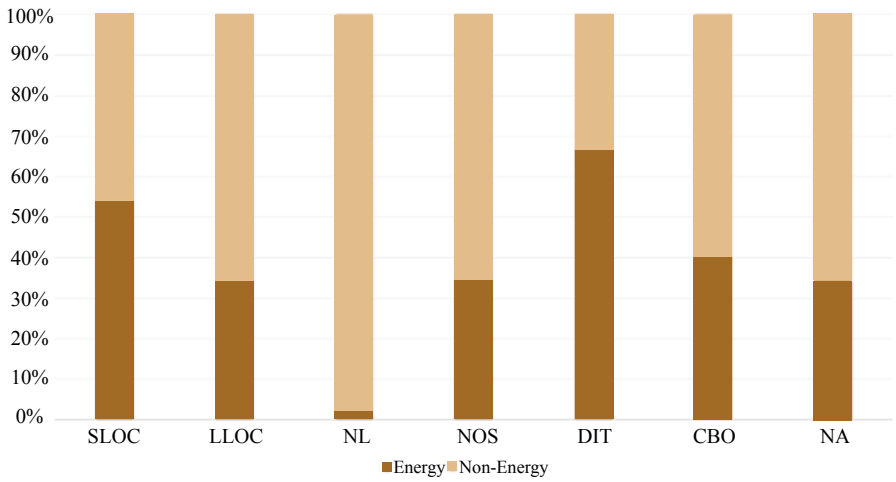


Figure 3.5: Source code metrics.

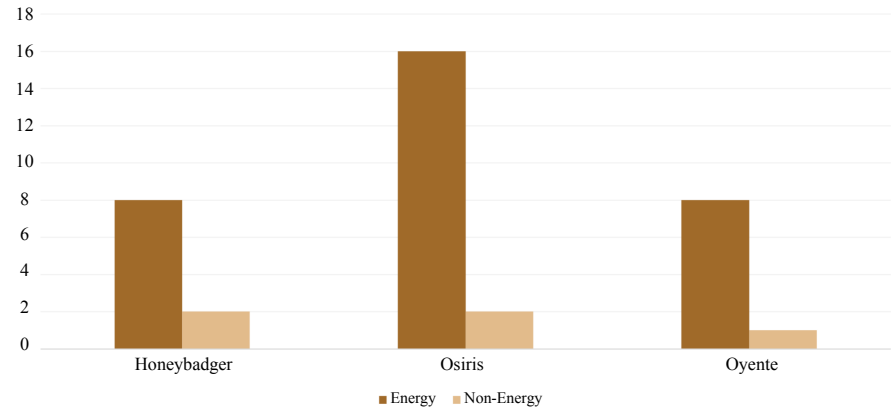


Figure 3.6: Contract analysis timeouts.

The dominant vulnerabilities across both categories of contracts are comparable, with a more ubiquitous presence in energy contracts, as illustrated in Figure 3.7. To determine whether there are domain-specific vulnerabilities or security flaws that emerge in particular application domains, Figure 3.8 summarizes the findings across all dominant vulnerabilities discovered for each category.

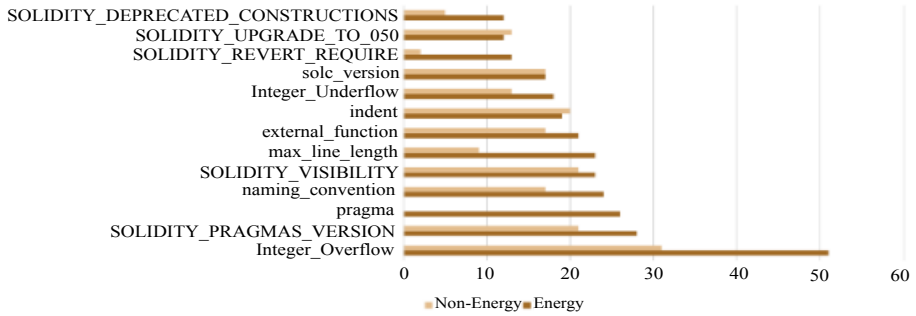


Figure 3.7: Dominant vulnerabilities.

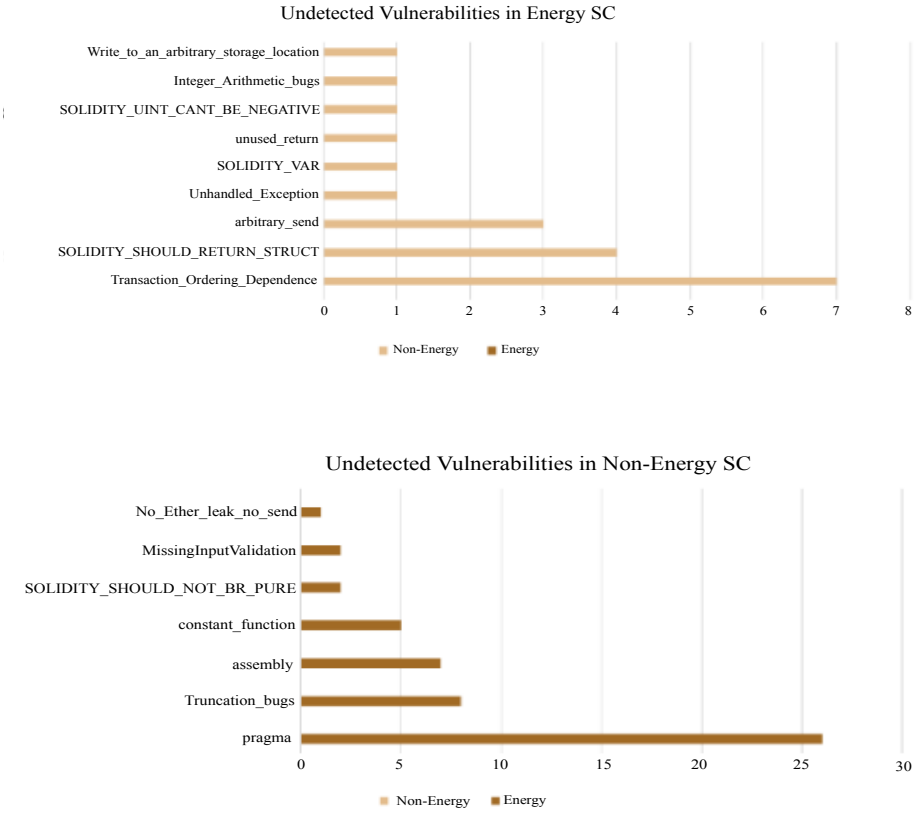


Figure 3.8: Undetected vulnerabilities.

There are comparable vulnerabilities among both categories of smart contracts. However, coding practices tend to be more detrimental among energy contracts due to the more prominent presence of each vulnerability in this domain.

In general, any vulnerability pertaining to an energy smart contract's logical statements could be exploited by malicious users to obtain unauthorized access,

manipulate energy prices, execute unauthorized transactions, or disrupt energy trading operations. Except for *solidity-revert-requires* and *solidity-deprecated-construction*, which are uncommon in non-energy contracts, the dominant vulnerabilities in both subgroups of contracts are mostly the same. However, *pragma* vulnerability, which involves the use of various pragma directives, was frequently observed in energy contracts as opposed to non-energy contracts.

Some vulnerabilities are prevalent in specific application domains, while others are extremely rare. Throughout the analysis, none of the designated tools detected a single instance of transaction ordering dependency in energy contracts. This observation is further reinforced by Figure 3.8, which shows the unidentified vulnerabilities in each category. However, energy smart contracts are built on complex economic models such as time-based conditions and real-time pricing and vulnerabilities affecting these models could have far-reaching implications. For instance, mishandling time-based conditions could result in incorrect pricing during peak or off-peak periods, potentially leading to financial losses for traders. This suggests that security flaws may be domain specific and can, thus, be overlooked by existing analytical tools that are not optimized for specific domains. Several studies reported that specific bugs, such as bad randomness, were completely overlooked by the benchmarked tools [37].

## 3.7 Discussion

According to empirical analytical studies of smart contracts, only a few of the security flaws revealed by benchmark tools are verifiable [161, 164]. Similar studies claim the evaluation results demonstrated numerous instances of vulnerabilities that fell within the tools' detection span but were not flagged by them. Additionally, there is no single tool that can detect all known vulnerabilities [28, 42, 109].

With a substantial number of false positives and false negatives, most of the existing vulnerability analysis tools do not meet the requirements of practical scenarios that rely on extensive manual verification.

As shown in Section 3.6, Slither demonstrated the highest accuracy for curated contracts. As a result, it could serve as a benchmark to assess the efficacy of other analysis tools on classified contracts, as suggested by previous vulnerability analysis research [169]. The following segment of this study is directed at validating this proposition.

Since the implications of reentrancy threat are primarily determined by the smart contract's business logic, the detection accuracy of the tools varies depending on the domain. In this section, we examine Slither's performance on classified contracts, pertaining to its effectiveness on reentrancy detection for curated smart contracts. There are a total of five and seven reentrant contracts detected in the energy and non-energy group, respectively.

According to the detection results in Table 3.2, Conkas detected reentrancy in all examined contracts, whereas Slither detected only one reentrancy in energy and three reentrancies in non-energy contracts. This can be interpreted as either False Positive (FP) for Conkas or False Negative (FN) for Slither. Therefore, we examined each contract independently to confirm the credibility of the results. The findings revealed Slither's inability to detect reentrancy, notably in energy contracts.

Table 3.2: Slither's reentrancy detection rate.

| Domain | Contract | Conkas | Slither | Osiris |
|---|---|---|---|---|
| **Energy** | Xad417b.sol | ✓ | × | × |
| | X3d9900.sol | ✓ | × | × |
| | X9001cb.sol | ✓ | × | × |
| | Xe69ba3.sol | ✓ | ✓ | × |
| | X5f10fd.sol | ✓ | × | × |
| **Non-Energy** | Xb99bf.sol | ✓ | ✓ | × |
| | XE3bcd.sol | ✓ | × | × |
| | X92658.sol | ✓ | × | × |
| | X6b0481.sol | ✓ | ✓ | × |
| | X326c72.sol | ✓ | × | × |
| | X75d9e6.sol | ✓ | × | × |
| | X091d3f.sol | ✓ | ✓ | ✓ |

The reentrancy vulnerability almost always results in the loss of smart contract money. However, in some cases, it may be the ability to create multiple instances of paid objects or to repeatedly orchestrate code that is only intended to be executed once per call [119, 174]. Generally, when exploiting a reentrancy vulnerability, the attacker aims to invoke the same contract function every time it is called. In programming, this is widely recognized as the recursion principle. Hence, most reentrancy attacks involve the *transfer*, *send* or *call* function. The *send* and *transfer* functions are considered slightly safer because they are restricted to 2300 gas and the gas constraint precludes the corresponding contract from making costly external function calls. Evidently, the *call* function is far more vulnerable [52, 67]. Whenever an external function call is expected to carry out complex operations, the *call* function is commonly employed to forward the remaining gas. This allows the attacker to resume the original function or a specific function from the original contract using a cross-function reentrancy.

For instance, looking at one of the energy contracts from Table 3.2, unlike Slither,

Conkas reports potential reentrancy on line 117, as shown in Figure 3.9. However, a closer look at line 3 in Listing 2, which equates to line 117 of the solidity file, points to a possible reentrancy. It may occur when the function *receiveApproval* directly or indirectly calls *approveAndCall*.

Listing 3.2: Smart contract with reentrancy.

```
1    function approveAndCall(address _spender, uint256 _value, bytes
        _extraData) returns (bool success) {
2        allowed[msg.sender][_spender] = _value;
3        Approval(msg.sender, _spender, _value);
4
5        if(!_spender.call(bytes4(bytes32(sha3("receiveApproval(
             address,uint256,address,bytes)"))), msg.sender, _value,
             this, _extraData)) { throw; }
6        return true;
```

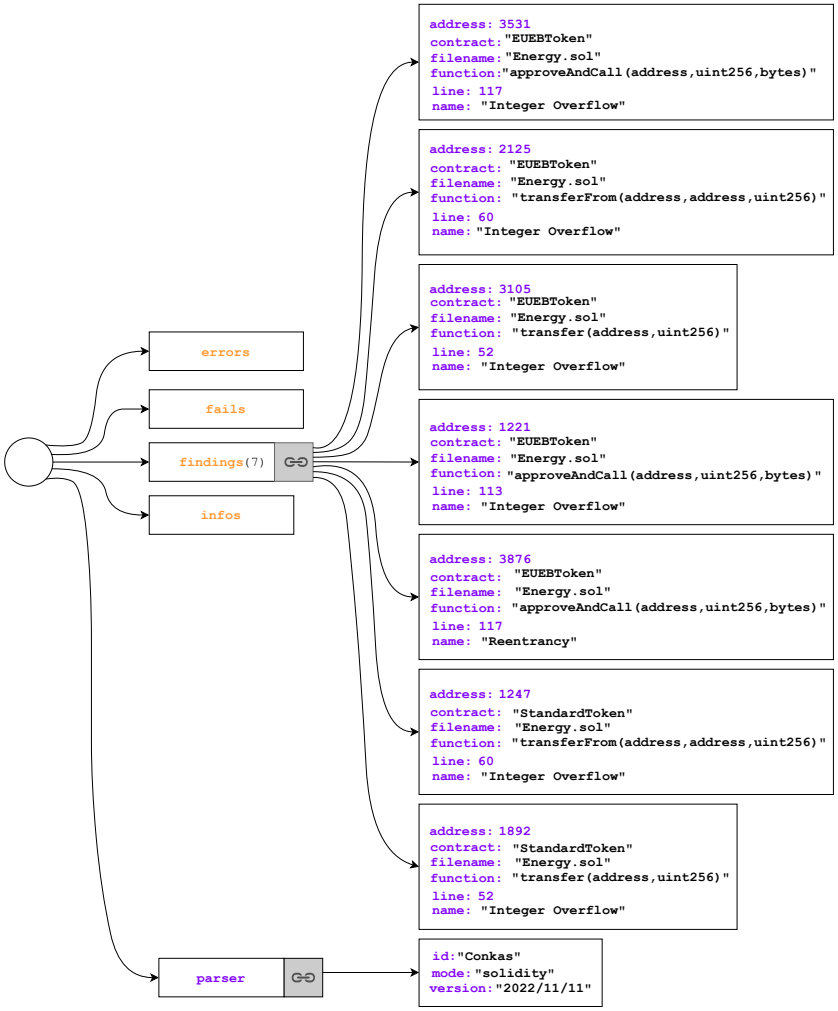


Figure 3.9: Conkas analysis results

This can be avoided by updating the state before making external calls, ensuring that the contract retains the most recent balance in the event that the attacker calls withdraw again. Although *call* is a low-level function for interacting with other contracts, it is not the preferred method for calling existing functions. It is strongly recommended to use *send* or *transfer* over *call* to reduce the attack surface [124, 125].

Another preventative measure to further improve the safety of contracts is to refactor the code. The objective of refactoring is to minimize the attack surface by adding pre- and post-conditions, such as *require* and *assert*, to encompass the vulnerable *call* invocations and ensure secure code structures. It ensures that all state variables are updated before any external calls to prevent the attacker from recursively calling the functions that are intended to be called once. The malicious node attempts to leverage the *call* and take over the control flow of the system by transmitting it to an external contract. By employing pre- and post-conditions, refactoring makes it possible to completely terminate a transaction if any errors emerge, protecting genuine users from potential risks.

Another form of refactoring is to safeguard the state of a contract with the inclusion of a state variable to manage mutual exclusion in functions. This approach is particularly beneficial when addressing cross-function reentrancy breaches. The primary objective is to protect code segments where common resources are accessed. Since mutual exclusion allows only one public function to be operating at a time, just one function will be able to modify the resource at once, completely eliminating cross-function reentrancy.

## 3.8 Conclusions

Smart contracts in the energy domain provide the necessary versatility to consolidate diverse processes according to the requirements of the application. Despite the positive association between smart contract categories and their vulnerabilities, vulnerability analysis tools do not consider violation structures and behavior patterns across

different application domains. Unfortunately, there is no perfect contract analysis tool for any and all contracts and their underlying business logic. Furthermore, the results of vulnerability analysis tools cannot be replicated in the absence of the appointed dataset. To examine this gap, the presented study evaluates the benchmark vulnerability analysis tools in classified and curated contracts.

This classification allows for an independent assessment of each domain's vulnerability source so that developers can differentiate between domain-specific vulnerabilities when paired with different execution environments. Although the vulnerability analysis workflow used in this research is only applicable to smart contract source code, it is transferable to any application domain in the presence of contracts in the corresponding domain. Therefore, it is reasonable to infer that the detection accuracy of the tools varies depending on the domain, since the benchmark tools did not demonstrate the accuracy claimed in the curated contracts. Furthermore, the overall impact of a comparable vulnerability in one application domain may be more profound and detrimental than in another. This makes the mitigation priority of these defects subject to business logic.

In addition, energy contracts demonstrate above-average security flaws and take longer to process, increasing the likelihood of failure or a timeout. The evaluation results revealed the competence of symbolic execution for the analysis of energy contracts. Accordingly, analysis tools that incorporate symbolic execution outperform code transformation coupled with constraint solving in detecting reentrancy in energy contracts.

Finally, the absence of granularity in governance regulation could lead to faulty smart contracts, sensitive endpoints, and ambiguous arbitration rules. One significant source of concern is the possibility of tampering, in which fraudulent parties exploit defects in smart contracts to gain unauthorized access or manipulate their intended functionality. One such attack vector is replay, where an attacker intercepts an authentic transaction and then replays it to illegitimately execute the same transaction. Therefore, a poor

governance approach may develop a secondary risk surface that can be exploited to launch a targeted attack on smart contracts. As a result, novel forms of vulnerabilities are emerging, such as cross-chain attacks, flashload attacks, and other types of attacks. Because smart contracts operate automatically based on predetermined conditions, this form of attack can have severe repercussions.

To reduce the risks associated with tampering, smart contract developers must proactively adhere to best practices. Additionally, given the growing interest in the adoption of smart contracts in areas such as metaverse, further investigation is required to ensure secure decentralized governance. Therefore, more efforts should be directed towards improving the reliability and accuracy of detection and disclosing new forms of vulnerability from a domain-specific point of view.

# Chapter 4

# Transfer Learning with Graph Neural Networks for Vulnerability Detection in Energy Smart Contracts

## 4.1 Abstract

The integration of smart contracts marks a significant advancement in peer-to-peer energy trading, offering a range of innovative and flexible features in this field. The inherent reliability and transparency of blockchain technology, combined with the automated execution of smart contracts, underpin a decentralized energy exchange system that allows consumers to participate in direct energy trading.

Despite their benefits, smart contracts, like any other software, are not immune to errors, vulnerabilities, and security issues, particularly when security patches are absent. When integrated into energy systems, smart contracts can create complex temporal and sequential dependencies, which can lead to a wide range of vulnerabilities as the structure of smart contracts becomes increasingly intricate. Current efforts to identify vulnerabilities in smart contracts rely heavily on expert-defined patterns, a method considered inefficient due to its lack of adaptability. Additionally, manual expert inspection is a time-consuming and resource-intensive process to collect sufficiently large and properly labeled datasets of smart contracts within specific domains.

To enhance the effectiveness of current methods, we introduce a graph attention

neural network model called TL-GAT, which leverages transfer learning to detect vulnerabilities in smart contracts. This framework enables developers to independently assess vulnerabilities in each domain, allowing them to accurately identify potential issues in diverse execution environments. The evaluation results confirm that transfer learning can be used to leverage existing knowledge of vulnerabilities to improve the effectiveness of generalization when addressing emerging vulnerabilities in a specific domain, even with limited data availability.

## 4.2 Introduction

The advent of blockchain as a distributed ledger offers a reliable decentralized execution scheme, which has fueled the rise of blockchain-powered platforms such as Ethereum [151], [131], [158]. The smart contract is an exceedingly important component of these platforms. It is executable code written in a high-level programming language that encapsulates the specifics of buyer-seller agreements and transaction regulations. Smart contracts offer several advantages, such as precision, efficacy, reliability, and transparency [101]. As a result, blockchain-based smart contracts have attracted considerable interest from various industries [35], [70]. The immutability and tamper resistance of these blockchain frameworks are additionally enforced on smart contracts, thereby ensuring that any terms documented in a smart contract cannot be altered once they have been published. However, smart contracts encounter security challenges, including an error-prone programming language with exploitable development bugs that are often overlooked or discovered only after deployment on the blockchain, making it difficult to fix them.

Smart contracts frequently govern substantial financial assets, making them attractive and easily targeted by malicious actors seeking financial gains. Bug exploitatioThe advent of blockchain as a distributed ledger offers a reliable decentralized execution scheme, which has fueled the rise of blockchain-powered platforms such as Ethereum [151], [131], [158].

The smart contract is an exceedingly important component of these platforms. It is executable code written in a high-level programming language that encapsulates the specifics of buyer-seller agreements and transaction regulations. Smart contracts offer several advantages, such as precision, efficacy, reliability, and transparency [101]. As a result, blockchain-based smart contracts have attracted considerable interest from various industries [35], [70]. The immutability and tamper resistance of these blockchain frameworks are additionally enforced on smart contracts, thereby ensuring that any terms documented in a smart contract cannot be altered once they have been published. However, smart contracts encounter security challenges, including an error-prone programming language with exploitable development bugs that are often overlooked or discovered only after deployment on the blockchain, making it difficult to fix them. Smart contracts frequently govern substantial financial assets, making them attractive and easily targeted by malicious actors seeking financial gains.

Bug exploitation in smart contracts can lead to severe consequences that affect the entire blockchain ecosystem, rather than just individual contracts. There have been several notable incidents caused by smart contract flaws, the most significant being the DAO incident, which was triggered by a reentrancy vulnerability that resulted in the unauthorized extraction of approximately $70 million in 2016 [175], [137]. As a result, identifying and detecting smart contract vulnerabilities has become a critical challenge that must be addressed promptly to mitigate potential financial losses caused by bug exploitation [111].

Several smart contracts vulnerability detection methods have been proposed to adapt classic software security strategies such as symbolic execution, data flow analysis, runtime monitoring, and fuzzing [86], [62].

The vast majority of existing smart contract vulnerability detection methods rely on static analysis to look for predefined vulnerability structures established by experts in target smart contracts [98]. Since these manually specified vulnerability structures are relatively simple, they are incapable of detecting intricate vulnerability patterns,

rely heavily on human expertise, and take an extended period to process smart contracts. For instance, while Conkas [121] previously demonstrated effectiveness in detecting a significant number of reentrancy vulnerabilities [77], it is unable to effectively identifying vulnerabilities within smart contract dependencies and library files and has reportedly logged over 300 false positive reentrancy detections across 31 samples [37], [99].

In addition, these detection techniques only search for specific vulnerabilities and are difficult to extend to new vulnerability types, as experts must examine emerging security defects to identify novel issues. With the rapid growth of smart contracts, it is becoming increasingly challenging to develope precise vulnerability structures with only a selected group of experts. Consequently, multiple detection tools need to be utilized to analyze diverse security flaws, thereby complicating testbed development for smart contract developeres and increasing processing overhead as well as runtime analysis.

Machine learning has recently garnered significant attention from security researchers owing to its ability to independently uncover hidden patterns in vast amounts of data [133], [106]. To avoid reliance on expert-defined structures, researchers have suggested various machine learning-based methods for detecting smart contract vulnerabilities. Several studies treat the smart contract as natural language, viewing it as a linear sequence, and then utilize a sequential neural network to perform feature extraction and vulnerability detection. Qian et al. [107] proposed the Bi-LSTM [136] with the attention mechanism to detect reentrancy by obtaining control flow and semantic information. DeeSCVHunter [165] proposed the concept of vulnerability candidate slices, which highlight fundamental vulnerability attributes, with an emphasis on reentrancy and time dependency. However, a smart contract consists of more intricate logic and structures than natural languages. As a result, the aforementioned methods overlook implicit semantic features involving data or control dependencies within the code's structure, leading to low precision in vulnerability detection.

Consequently, a number of graph-based methods have been proposed to safeguard the non-sequential semantic features of smart contract source code [13]. Zhuang et al. [179] investigated the detection of smart contract vulnerabilities using a graph neural network (GNN) and contract graph. Each graph portrays the syntactic and semantic patterns for a smart contract function and is used in combination with a degree-free graph convolutional neural network (DB-GCN) [162] and a novel Temporal Messaging Network (TMN) [178] for vulnerability detection. However, similar to DB-GCN, the majority of graph-based techniques employ a unidirectional graph convolutional network to obtain the distinct characteristics of a graph. This generates the node feature as well as each node's outgoing edges, which prevents the model from discovering the contextual properties of each node over inbound edges. The effectiveness of these methods depends on the availability of well-structured training data, which can only be acquired through manual audits performed by experts. In addition, preceding studies have shown that the performance of comparable vulnerability analysis tools is significantly different across various smart contract application domains.

Despite reports indicating a positive association between smart contract categories and their underlying defects, vulnerability analysis tools have yet to consider the application domain or purpose of the contracts under evaluation [60], [77]. Collectively, existing methods impose limitations on the ability to explore newly emerging vulnerabilities and advance domain-specific vulnerability analysis.

To address the problem of limited domain adaptation in existing approaches, the primary objective of this research is to introduce transfer learning in the detection of vulnerabilities in smart contracts. By harnessing knowledge from one domain, we aim to enhance the model's effectiveness in a new domain [17]. This approach has the capacity to adapt to emerging vulnerabilities and patterns through fine-tuning, thereby minimizing the requirement for substantial labeling within the target domain. To examine this concept, in this study the vulnerability analysis is carried out on smart contracts deployed in transactive energy systems.

In a previous study examining smart contracts [77], we noted that while the source code demographic profiles suggested higher values for metrics such as the number of logical lines of code (LLOC), depth of nesting levels (NL), and the number of attributes (NA) and statements (NOS) in non-energy contracts, the energy contracts exhibited a relatively greater number of defects when compared across an equal number of contracts in both categories. However, the process of obtaining sufficient and properly labeled smart contracts within specific domains involves the resource-intensive and time-consuming process of manual expert inspection. Nevertheless, training of the Graph Neural Network (GNN) within a specific domain evidently improves the model's learning performance, resulting in more refined vulnerability detection results.

In this study, we propose a GNN-based model that leverages the structural information of smart contracts encoded as graphs for the detection of reentrancy attacks [177], [163]. Accordingly, in this study we propose Transfer Learning with Graph Attention Networks (TL-GAT) to train the GNN on a wide range of smart contracts from various domains, serving as the source domain. Following that, we fine-tune the model to detect reentrancy within the domain of energy smart contracts, which is our target domain. The proposed method leverages the knowledge of the pre-trained model, tailoring it to detect vulnerabilities associated with reentrant data points as they manifest in the energy sector, with the goal of achieving the highest detection accuracy compared to all previous efforts.

## 4.3 Background

Along with the continuing trend toward a decentralized and dynamic energy landscape, the viability of blockchain-powered smart contracts in transactive energy systems has garnered significant interest. The integration of smart contracts has emerged as a significant advancement in the context of peer-to-peer (P2P) energy trading [110], [69]. Smart contracts enable a wide range of innovative and dynamic functions in energy trading, such as auction mechanisms and the enforcement of pricing policies

and exchanges [100].

The inherent reliability and transparency of blockchain technology, combined with the automated properties of smart contracts, contribute to a decentralized energy exchange system that enables consumers to participate in direct energy trading. Through the use of peer-to-peer transmission smart contracts, participants can safely place offers and purchase energy in the decentralized market. This way, consumers can receive a proper offer from the various suppliers [23]. Considering the array of features they encompass, smart contracts implemented within transactive energy systems introduce highly intricate time and sequencing dependencies. As a result of the increasing complexity of smart contract structures, a broad range of vulnerabilities are developed [26], [15], [66].

Hence, it is critical to ensure that a smart contract is thoroughly examined to look for potential encoded programming defects that might result in serious vulnerabilities. An attacker with malicious intent could exploit these defects to trigger unintended processes that the smart contract was not originally designed to perform. As a result, it is necessary to identify vulnerabilities in smart contracts prior to their deployment [71]. Furthermore, the implications of a comparable vulnerability within one application domain may be more severe and destructive than in another. Consequently, the order in which these vulnerabilities are addressed, as well as their detection process, is determined by business logic.

The focus of this study is on the detection of reentrancy vulnerability to mitigate the implications of reentrancy attacks in transactive energy systems [34]. The reentrancy attack is a common exploit arrangement that uses external calls to deplete a smart contract [174]. This attack strategy, shown in Fig.4.1, employs an optimized *fallback()* function to iteratively invoke *withdraw()* leading up to updating the user's balance. The critical part of this instance occurs when *MaliciousContract* is called to initiate a reentrancy attack. The *attack()* function in this contract is called, which contains the *withdrawEnergy()* function in the *EnergyContract*. This function is originally

Vulnerable Contract

```
contract EnergyContract {
    mapping(address => uint256) public energyBalances;

    function depositEnergy(uint256 _amount) public {
        energyBalances[msg.sender] += _amount;
    }

    function withdrawEnergy(uint256 _amount) public {
        require(energyBalances[msg.sender] >= _amount, "Insufficient
        energy balance");

        // Simulate energy transfer
        energyBalances[msg.sender] -= _amount;
        // In a real contract, there would be logic to transfer energy
        to the recipient.

        // Simulate a callback to an external contract
        MaliciousContract malicious = MaliciousContract(msg.sender);
        (bool success, ) = address(malicious).call{value: 0 ether}("");
        require(success, "Reentrancy attack failed");
    }
}
```

Malicious Contract

```
contract MaliciousContract {
    EnergyContract public energyContract;

    constructor(address _energyContractAddress) {
    energyContract = EnergyContract(_energyContractAddress);
    }

    function attack() public {
    // Perform a reentrancy attack
    energyContract.withdrawEnergy(1);
    }

    // Fallback function to receive Ether
    receive() external payable {

        if (msg.sender != address(energyContract)) {
            // Prevent reentrancy attack from other sources
            attack();
        }
    }
}
```

Figure 4.1: Reentrancy exploitation instance

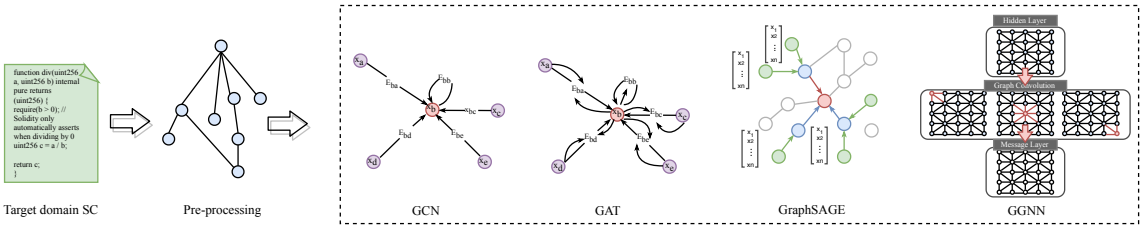


Figure 4.2: Proposed Vulnerability Detection Workflow

meant to update the energy balance. Since the *MaliciouseContract* leverages this function to call back into the *EnergyContract*, the *recieve()* function is triggered. As a result, the *attack()* function is invoked once again, creating a loop. Therefore, the attacker can repeatedly call the *attack()* function, draining the energy balance of the *EnergyContract* reentrantly, resulting in a successful reentrancy attack. In summary, reentrant energy contracts can lead to inaccurate energy allocations and financial losses for parties and ultimately impact the stability and dependability of the grid, potentially resulting in power outages [112]. In the following section, we propose a framework for the effective detection of this defect using graph neural networks and transfer learning.

## 4.4 Methodology

When analyzing software processes such as smart contracts for potential vulnerabilities, it is helpful to identify the relationships between their various components. As solidity is the most widely used programming language for implementing smart contracts, much research on detecting vulnerabilities focuses on examining the source code of these contracts. Following a similar approach, we have chosen to use the source code in our research to enhance the practicality of performing comprehensive analysis and assessment.

In the proposed approach, graph representations for each smart contract are generated to derive adequate syntax and semantic information. As a result, solidity code files undergo a prepossessing phase to enable graph-based representation. As illustrated in Fig. 4.2, after preprocessing, the smart contracts' graph representations are fed into an array of graph neural network techniques for training and predicting potential reentrant data points. The subsequent sections thoroughly describe the workflow for vulnerability detection.

### 4.4.1 Pre-processing

The prepossessing pipeline contains solidity code parsing, which tokenizes and analyzes the source code to identify different components. To identify distinct elements in the source code, the parser implements a set of regular expression patterns. These patterns enable the parser to identify components such as function declarations, function calls, event declarations, etc. The parser then employs a dictionary known as node-type features to characterize these different elements inside the graph. This dictionary plays an integral role in mapping various types of nodes to distinct numerical feature vectors. The vectors that follow become identifiers, allowing the parser to distinguish all of the different types of components that exist in the code. Edges develop between the parent contract node and the functions and events associated with it. Each node in the

graph has been assigned a type based on its purpose in the source code, and the edges linking these nodes demonstrate their interdependence. These edges demonstrate the connection between the contract and its internal elements.

Finally, the pipeline's output is a systematically structured graph representation of the solidity smart contracts. This intricate representation facilitates an understanding of how the contract's components interact, providing greater insight into the complexities of smart contract programming [129]. Data processing is carried out on a dataset of 4,756 Solidity smart contracts, labeled for reentrancy vulnerbility [84].

### 4.4.2 Graph Convolutional Network Model

Graph Convolutional Network (GCN) is essentially comparable to a Convolutional Neural Network (CNN), in that it acquires domain-specific information for processing. However, the primary distinction is the applicability of the GCN to data with non-Euclidean structure [172], [81].

The GCN model seamlessly incorporates the connection sequences and features that are inherent in graph-structured data, outperforming numerous state-of-the-art techniques in particular domains. GCN's architecture consists of a graph convolution layer, a graph readout layer, a graph regularization layer, and a graph pooling layer to boost generalization while minimizing the number of computational parameters. The foundational equation of the GCN can be written as follows

$$H_{l+1} = \sigma(D^{-\frac{1}{2}}\tilde{A}D^{-\frac{1}{2}}H_l w_l), \tag{1}$$

where $H_{l+1}$ is the output at layer $l+1$, $\sigma$ is the activation function, $D^{-\frac{1}{2}}$ and $D^{-\frac{1}{2}}$ stand for the diagonal degree matrices, $\tilde{A}$ symbolizes the normalized adjacency matrix, $H_1$ corresponds to the input at layer $l$, and $w_l$ is weight matrix at layer $l$. Messages are aggregated from neighbouring nodes and current node embeddings $H(l-1)$. The aggregated messages then undergo a linear transformation using learnable weights. The ReLU activation function is used in the hidden layers to introduce nonlinear

properties, whereas the Sigmoid activation function is used in the output layer ($l = L$) for binary classification. This prevents overfitting by introducing a certain degree of regularization and constrains the model's ability to fit noise in the data, assisting generalization to previously unseen examples.

As shown in Algorithm 1, the GCN model adapted for vulnerability detection iteratively updates the node embeddings in a graph by aggregating and refining the information from neighboring nodes using learnable weights. This allows effective vulnerability detection in graphs corresponding to smart contracts.

The network architecture is composed of an input layer, an output layer, and a number of hidden layers, each of which passes its output through the activation function. The output layer then releases a prediction label after multiple layers of computation. The objective of the model is to correctly predict the label $\hat{y}$, where a value of 1 implies the presence of a reentrancy and any other value suggests that the smart contract is safe.

**Algorithm 1** Graph Convolutional Network (GCN) for Vulnerability Detection

**Input:** Graph $G = (V, E)$, node features $X(v_i) \in \mathbb{R}^d$, adjacency matrix $\mathbf{A}$
**Initialization:** Learnable parameters $\Theta$
**Output:** Predicted vulnerability labels $\hat{y}(v_i)$ for all nodes $v_i \in V$
Compute normalized adjacency matrix: $\hat{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ ▷ Where $\mathbf{D}$ is the degree matrix of $\mathbf{A}$
Initialize node embeddings: $H^{(0)} = X$
**for** $k = 1$ to $K$ **do** ▷ Number of layers
    **Message Aggregation:**
    Compute message vectors: $M^{(k)} = \hat{\mathbf{A}}H^{(k-1)}$
    **Message Transformation:**
    Apply linear transformation: $Z^{(k)} = M^{(k)}W^{(k)}$ ▷ Where $W^{(k)}$ is a learnable weight matrix
    **Aggregation:**
    Aggregate messages: $H^{(k)} = \text{ReLU}(Z^{(k)})$
    ▷ ReLU activation function
**end for**
**Output:** $\hat{y}(v_i) = \sigma(H^{(K)}(v_i))$ for all nodes $v_i \in V$ ▷ Apply sigmoid activation function

### 4.4.3 Graph Attention Convolutional Network Model

The model is made up of two layers of GATConv (Graph Attention Network), which are mainly intended for processing graph-structured data [87].

The model is trained using labeled data, each instance of smart contract assigned a ground truth label of re-entrant or non-re-entrant. Each GATConv layer is in charge of processing information from the graph's nodes and edges while introducing the attention mechanism as shown in equation 2. $X_i$ represents the feature vector of node $i$ in the graph, $W$ is a learnable weight matrix, and $a$ represents the attention weights. Afterwards, the attention coefficients are calculated using softmax in equation 3 where $e_{ij}$ is used to demonstrate the unnormalized attention score between nodes $i$ and $j$. Accordingly, $\alpha_{ij}$ corresponds to the attention coefficient for the edge between nodes $i$ and $j$ while $N_i$ represents the neighbors of node $i$ and $h'_i$ returns the updated representation of node $i$.

$$e_{ij} = \text{ReLU}(a^T \cdot [W \cdot X_i || W \cdot X_j]) \tag{2}$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in N_i} \exp(e_{ik})} \tag{3}$$

$$h'_i = \sum_{j \in N_i} \alpha_{ij} \cdot W \cdot X_j \tag{4}$$

This attention procedure enables the model to allocate different levels of importance to neighboring nodes through message passing, making GATConv an effective method for analyzing graph-structured data. The first GATConv layer handles the input node features and edge indices generated during data preprocessing. Four attention heads are used in this layer, each of which aggregates information independently, ensuring that the model captures varying connections within the graph. The use of attention heads allows for concurrent processing and aggregation of various node-level information, which contributes to more reliable vulnerability projections. The resulting node embeddings preserve graph-level information while taking node interactions into

account.

Following the attention mechanisms, a Rectified Linear Unit (ReLU) activation function is implemented to inject nonlinearity into the model, allowing it to identify intricate patterns in the data. Additionally, GATConv reinforces message passing, in which each node aggregates knowledge obtained from its neighbors based on attention scores. The outputs from the first layer are directly aggregated by the second GATConv layer. The final output of the second GATConv layer pertains to node embeddings that capture key attributes and connections in the smart contract graph. These embeddings are used to make projections about the smart contract's reentrancy status. The sigmoid activation function *sigma* is then applied to the dot product of learnable weights $W_v$ and node embeddings to predict vulnerability state for each node.

### 4.4.4 GraphSAGE

GraphSAGE [49] develops embeddings for previously unseen data points by sampling and capturing features from adjacent neighborhoods in each layer. Unlike traditional methods that train separate embeddings for each node using matrix factorization, GraphSAGE employs a function that generates embeddings by aggregating features from a node's local neighborhood. By integrating node attributes, GraphSAGE generates both the topological structure of a node's surroundings and the patterns of attribute distribution within that area.

As shown in Algorithm 3, instead of training individual embedding vectors for each node, a set of aggregator functions is created to efficiently combine attributes from a node's immediate vicinity. The basic premise of this algorithm is that the nodes obtain information from their neighboring nodes at each interval. As this procedure is amplified over the course of iterations, the nodes gradually accumulate additional details from deeper fragments of the graph.

The final output consists the node embeddings $h^{(L)}(v_i)$, which encapsulate the acquired node representations, and the projection of vulnerability $\hat{y}(v_i)$, which indicate

**Algorithm 2** Graph Attention Network (GAT) for Vulnerability Detection

**Input:** Node features $X \in \mathbb{R}^{N \times D}$, edge indices $E \in \mathbb{N}^{2 \times M}$.
**Output:** Vulnerability predictions $\hat{y} \in \mathbb{R}^{N}$.
Initialize parameters: Input dimension $D$, hidden dimension $H$, output dimension $F$.
Initialize learnable weight matrices: $\mathbf{W}^{Q} \in \mathbb{R}^{H \times H}$, $\mathbf{W}^{K} \in \mathbb{R}^{H \times H}$, $\mathbf{W}^{V} \in \mathbb{R}^{H \times H}$.
Initialize self-attention mechanism parameters: Number of attention heads $K$, dropout rate $p$.
Initialize aggregation function: Concatenation.
Define the graph attention mechanism:
**function** GraphAttention($X, \mathbf{W}^{Q}, \mathbf{W}^{K}, \mathbf{W}^{V}, K, p$)
    Compute query, key, and value matrices: $\mathbf{Q} = X\mathbf{W}^{Q}$, $\mathbf{K} = X\mathbf{W}^{K}$, $\mathbf{V} = X\mathbf{W}^{V}$.
    Split matrices into $K$ heads: $\mathbf{Q}_k, \mathbf{K}_k, \mathbf{V}_k$ for $k = 1$ to $K$.
    Calculate attention scores: $\text{Attention}(\mathbf{Q}_k, \mathbf{K}_k) = \frac{\exp(\mathbf{Q}_k \mathbf{K}_k^{\top})}{\sum_{j=1}^{N} \exp(\mathbf{Q}_k \mathbf{K}_k^{\top})}$.
    Apply dropout to attention: $\text{Attention}(\mathbf{Q}_k, \mathbf{K}_k) \leftarrow \text{Dropout}(\text{Attention}(\mathbf{Q}_k, \mathbf{K}_k), p)$.
    Compute output for each head: $\mathbf{Z}_k = \text{Attention}(\mathbf{Q}_k, \mathbf{K}_k)\mathbf{V}_k$.
    Concatenate head outputs: $\mathbf{Z} = \text{Concatenate}(\mathbf{Z}_1, \mathbf{Z}_2, \ldots, \mathbf{Z}_K)$.
    Return the output feature matrix $\mathbf{Z}$.
**end function**
Define the GAT model:
**function** GAT($X, E$)
    Initialize node embeddings: $\mathbf{H} = X$.
    Apply graph attention mechanism $\mathbf{H}$: $\mathbf{H} = \text{GraphAttention}(\mathbf{H}, \mathbf{W}^{Q}, \mathbf{W}^{K}, \mathbf{W}^{V}, K, p)$.
    Apply activation function: $\mathbf{H} = \text{ReLU}(\mathbf{H})$.
    Apply dropout: $\mathbf{H} = \text{Dropout}(\mathbf{H}, p)$.
    Return node embeddings $\mathbf{H}$.
**end function**
Initialize the GAT model: $G \leftarrow \text{GAT}(X, E)$.
**Inference:**
Use the trained GAT model to predict vulnerability scores $\hat{y}$, where $\hat{y}_i = \sigma(W_v \cdot \mathbf{H}_i)$ for each node $i$. ▷ Where $\sigma$ is the sigmoid activation function and $W_v$ are learnable weights.

the predicted label of each node exhibiting reentrancy.

**Algorithm 3** GraphSAGE Algorithm for Vulnerability Detection

**Input:** Graph $G = (V, E)$, node features $X(v_i) \in \mathbb{R}^d$, number of layers $L$
**Initialization:** Learnable parameters $\Theta$
**Output:** Node embeddings $h^{(L)}(v_i) \in \mathbb{R}^d$ and predicted vulnerability labels $\hat{y}(v_i)$ for all $v_i \in V$
**Initialization:**
$h^{(0)}(v_i) \leftarrow X(v_i)$ for all $v_i \in V$
**for** $l = 1$ to $L$ **do**
    **Aggregation at Layer** $l$**:**
    $Z^{(l)}(v_i) \leftarrow \text{Aggregator}^{(l)}\left(h^{(l-1)}(v_i), \text{neighborhood samples}\right)$
    **Activation Function at Layer** $l$**:**
    $h^{(l)}(v_i) \leftarrow \text{Activation}\left(Z^{(l)}(v_i)\right)$
    **Update Node Representations:**
    $h^{(l+1)}(v_i) \leftarrow h^{(l)}(v_i)$ for all $v_i \in V$
**end for**
**Vulnerability Detection:**
$\hat{y}(v_i) = \sigma(f(h^{(L)}(v_i)))$ for all $v_i \in V$ ▷ Apply sigmoid activation function
**Output:**
Node embeddings $h^{(L)}(v_i)$ and predicted vulnerability labels $\hat{y}(v_i)$ for all $v_i \in V$

### 4.4.5 GGNN

Gated Graph Neural Networks (GG-NNs) are an instance of GNNs that can learn representations at both the node level and the graph level [80]. GGNN uses Gated Recurrent Units and extends the recurrence over a fixed number of iterations $T$. It also employs backpropagation to derive gradients.

In GGNN, the GRU-based message transmission process involves receiving features from neighbors, aggregating these properties, and revising node representations. As shown in Algorithm 4, GNN is provided with input in the form of a graph $G = (V, E)$, accompanied by node features $X(v_i)$ and edge features $E(v_i, v_j)$. It then begins with the initialization of the learnable parameters denoted as $\Theta$.

In each iteration ($t = 1 \ldots T$), the algorithm processes each node $v_i$ in the graph sequentially. It begins by establishing the initial hidden state $h_t(v_i)$ with the node's respective feature $X(v_i)$. For each neighboring node $v_j$ of $v_i$, a message $m_t(v_i, v_j)$ is

computed, implementing a ReLU activation function. Subsequently, these individual messages are aggregated to generate $m_t(v_i)$. A sigmoid activation function, gating procedure along with a combination of the current hidden state $h_t(v_i)$ and the aggregated message $m_t(v_i)$ are used to determine the gate values $g_t(v_i)$ for each node $v_i$. The algorithm then proceeds to vulnerability detection after (T) iterations. It determines reentrancy labels $haty(v_i)$ for all nodes $v_i$ using sigmoid activation of the ultimate hidden state $h_T(v_i)$.

**Algorithm 4** Gated Graph Neural Network (GGNN) Algorithm for Vulnerability Detection

**Input:** Graph $G = (V, E)$, node features $X(v_i) \in \mathbb{R}^d$, edge features $E(v_i, v_j) \in \mathbb{R}^d$
**Initialization:** Learnable parameters $\Theta$
**Output:** Updated node features $X'(v_i) \in \mathbb{R}^d$ and predicted vulnerability labels $\hat{y}(v_i)$ for all nodes $v_i \in V$
**for** $t = 1$ to $T$ **do** ▷ Number of iterations
    **for** each node $v_i \in V$ **do**
        **Message Aggregation:**
        Initialize hidden state $h_t(v_i) = X(v_i)$
        **for** each neighbor $v_j$ of $v_i$ **do**
            Compute message $m_t(v_i, v_j) = \text{ReLU}(E(v_i, v_j) \cdot h_t(v_j))$
            Aggregate messages: $m_t(v_i) = \sum_{v_j} m_t(v_i, v_j)$
        **end for**
        **Gating Mechanism:**
        Compute gate $g_t(v_i) = \sigma(W_g \cdot [h_t(v_i), m_t(v_i)])$
        **Update Node State:**
        Compute updated state: $s_t(v_i) = \text{ReLU}(W_s \cdot [h_t(v_i), g_t(v_i) \cdot m_t(v_i)])$
        **Update Hidden State:**
        Update hidden state: $h_{t+1}(v_i) = s_t(v_i)$
    **end for**
**end for**
**Vulnerability Detection:**
$\hat{y}(v_i) = \sigma(f(h_T(v_i)))$ for all $v_i \in V$ ▷ Apply sigmoid activation
**Output:** $X'(v_i)$ and $\hat{y}(v_i)$ for all nodes $v_i \in V$

### 4.4.6 TL-GAT

To develop the transfer learning framework, we use a Graph Neural Network (GNN) model to detect vulnerabilities in the source domain [163]. The idea behind the

TL-GAT is to use transfer learning to migrate the previously trained GAT model to new datasets. Transfer learning allows the use of existing vulnerability knowledge to improve the generalization of emerging vulnerabilities with limited data [17], [113].

TL-GAT's transfer learning capacity allows our detection framework to be improved at a low cost to safeguard against emerging smart contract defects. The proposed transfer learning mechanism allows for the preservation of knowledge on preceding expressions of vulnerabilities during the initial training phase. Additionally, TL-GAT is intended to effectively adapt the pre-trained model to obtain maximum precision on the new task data with limited size. This is especially important as training a GNN model from scratch for unexplored expressions of vulnerabilities across various domains with limited data is prohibitively expensive and challenging to maintain.

As depicted in Fig.4.3, the initial phase involves training on the source domain. During this phase, we train the GAT model using source codes spanning across different application domains because it is more feasible to gather a substantial amount of vulnerabilities, particularly reentrancy expressions, using cross-domain smart contracts. While these vulnerabilities are not domain-specific, in line with the principles of transfer learning, TL-GAT has the ability to obtain valuable knowledge during this stage that can be leveraged for improved vulnerability detection. Following the initial source domain training, the second phase involves the target domain training.

Therefore, the pre-trained model can be fine-tuned on the target domain dataset, and its effectiveness can be evaluated. During the second phase, we leveraged a high-quality dataset comprising energy smart contracts labeled with reentrancy vulnerabilities. Although the dataset exclusively focuses on reentrancy vulnerabilities, its main constraint is its limited size. This limitation arises from the scarcity of audited energy smart contracts screened for security defects. Labeling these energy smart contracts is particularly challenging due to the significant time and expense involved in manual auditing. A new optimizer is developed to fine-tune the model on a target domain dataset and transfer knowledge acquired from cross-domain defects. As we

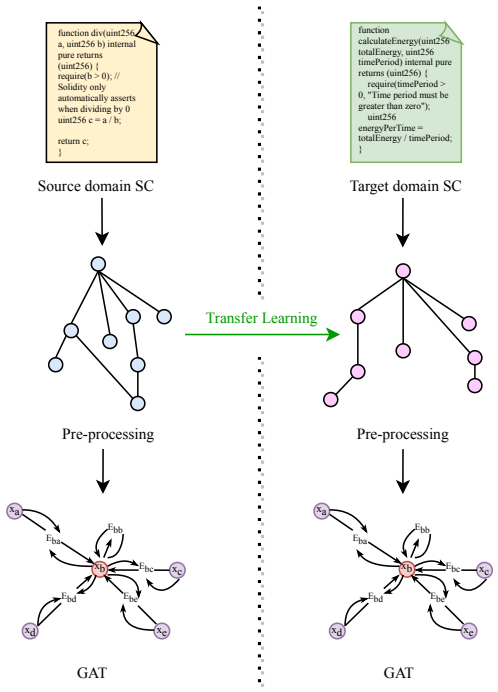


Figure 4.3: Transfer Learning with GAT

will see later in Section 4, TL-GAT outperforms other proposed GNNs when trained exclusively on energy smart contracts.

## 4.5 Results and Discussion

In this section, we evaluate the performance of the proposed framework for each model separately. The effectiveness of each model for reentrancy is assessed using Accuracy, Recall, Precision, and F1-score, which are reported in Table 4.1.

In turn, an assessment is conducted on True Positive Rate (TPR), False Positive Rate (FPR), True Negative Rate (TNR), and False Negative Rate (FNR), as shown in Fig. 4.4. Among all the models proposed in the framework, GAT with Transfer Learning achieved the highest performance, with 99.06% accuracy, 98.50 % precision, 100% recall, and 99.25% F1-score. Accuracy examines the proportion of accurately predicted instances to the total number of samples. The competency of the model to identify the vulnerable instances is measured by recall, using the ratio of true

**Algorithm 5** Transfer Learning with GAT Algorithm for Vulnerability Detection

**Input for Source Domain:** Graph $G_S = (V_S, E_S)$, node features $X_S(v_i) \in \mathbb{R}^d$, labels $Y_S(v_i)$ for source domain
**Input for Target Domain:** Graph $G_T = (V_T, E_T)$, node features $X_T(v_i) \in \mathbb{R}^d$ for the target domain
**Pretrained GAT Model:** GAT model with parameters $\Theta$ trained on $G_S$ for node classification
**Output:** Node embeddings $H_T(v_i) \in \mathbb{R}^d$ for the target domain, Predicted vulnerability labels $y_T(v_i)$ for nodes in the target domain
**Initialization:**
Load pretrained GAT model with parameters $\Theta$
**Fine-tuning on Target Domain:**
Freeze all layers of the pretrained GAT model except the last layer
**Loss Function:** Cross-Entropy Loss
**Vulnerability Detection Loss Function:** Binary Cross-Entropy Loss for Vulnerability Detection
**for** $epoch = 1$ to $N$ **do**
    Sample a mini-batch of nodes $\{v_i\}$ from $G_T$
    Calculate node embeddings $H_T(v_i)$ using the pretrained GAT model
    Compute node classification loss on $\{v_i\}$ using $H_T(v_i)$ and $Y_S(v_i)$
    Compute vulnerability detection loss on $\{v_i\}$ using $H_T(v_i)$ and $y_T(v_i)$
    Update model parameters $\Theta$ using backpropagation
**end for**
**Output:** $H_T$ contains node embeddings for the target domain with updated model parameters for vulnerability detection, $y_T$ contains predicted vulnerability labels for nodes in the target domain

reentrancy projections in relation to all reentrant instances. Precision calculates the ratio of correctly identified vulnerable samples to the total number of vulnerable predictions including false positives. Consequently, the harmonic average of precision and recall is used to calculate the F1 score. In addition, false positives and false negatives pose major implications in vulnerability detection of smart contracts, since they disrupt the dependability, security, and effectiveness of the detection procedures. False positives occur if the detection method incorrectly flags a code segment as vulnerable. This causes unnecessary alerts, taking up time and resources exploring non-existent vulnerabilities. Recurring false positives may undermine the credibility of the detection system over time, resulting in a disregard or skepticism about the actual vulnerabilities reported by the system. On the other hand, false negatives appear when the detection framework overlooks an actual vulnerability, exposing the system to exploitation of defects. The frequent occurrence of false negatives decreases the reliability of the detection framework, possibly leading to exploitations and security breaches. This superiority can be attributed to its ability to leverage knowledge from a pre-trained source domain, which improves its overall predictive capacity. With recall, TL-GAT has a remarkable ability to capture a significant number of vulnerabilities. This observation is further reinforced by examining the TPR and FNR values specifically related to TL-GAT. This is especially important when dealing with reentrancy where a single reentrant point can have serious security implications.

Additionally, fine-tuning GAT on the target domain allows TL-GAT to better distinguish between true positives and effectively minimize false positives. The evaluation results show that even the most effective GNN variants, such as GCN, GAT, GGNN, and GraphSAGE, struggle with domain adaptation. Particularly due to the challenges associated with obtaining a substantial amount of labeled vulnerable contracts, these models become less viable, underscoring their limited practicality in situations with limited data availability, such as transactive energy systems. This issue is especially significant in the context of energy smart contracts, as they may

| Method | Accuracy(%) | Precision(%) | Recall(%) | F1(%) |
|---|---|---|---|---|
| GAT | 85.98 | 81.84 | 100 | 89.79 |
| GCN | 71.21 | 70.31 | 100 | 82.56 |
| GGNN | 92.95 | 89.75 | 100 | 94.62 |
| GraphSAGE | 77.46 | 91.17 | 70.45 | 79.48 |
| TL-GAT | 99.06 | 98.50 | 100 | 99.25 |

Table 4.1: Vulnerability detection performance of each method

involve undiscovered structures or vulnerability patterns.

Models that lack the ability to generalize may struggle to detect vulnerabilities that fall outside their training scope. Unlike TL-GAT, which leverages prior knowledge from a related source domain, the other examined models typically lack this capability. This implies that they might overlook valuable patterns and trends that could be transferred from one domain to another. Without the capacity for effective adaptation and generalization, these models could be prone to overfitting when trained on limited data. Overfitting can lead to poor generalization and inaccurate predictions on unseen smart contract codes. As illustrated in Table 4.1, GAT, GCN, and GGNN achieve significantly higher recall rates at the cost of more false positives.

In conclusion, the comparative advantage of GAT with Transfer Learning in domain-specific analysis for smart contract vulnerability detection can be attributed to its data efficiency, generalization, and domain-adaptive characteristics. TL-GAT effectively addresses the issue of limited labeled data while maximizing accuracy and precision. In the dynamic landscape of smart contract vulnerabilities, the ability of transfer learning to continuously adapt and learn makes it a valuable tool for enhancing security and dependability in this domain.

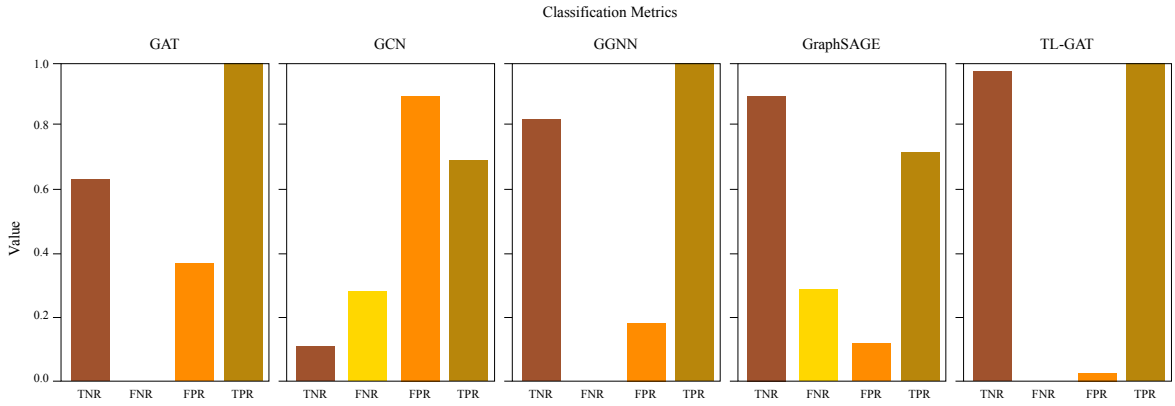


Figure 4.4: Evaluation results

## 4.6 Conclusion

The reliability of smart contracts is critical for the sustained growth of blockchain technology. In an effort to enhance existing methods, we introduce a graph attention neural network model based on transfer learning to detect vulnerabilities in smart contracts. This framework enables independent assessment of vulnerability sources in each domain, allowing developers to distinguish domain-specific vulnerabilities, especially when these contracts are used in diverse execution environments.

The premise of TL-GAT is to employ transfer learning to adapt a pre-trained GAT model for application to a new domain. Transfer learning enables the utilization of previously acquired knowledge about vulnerabilities to enhance the generalization of emerging vulnerabilities, even when dealing with limited data. This is of particular importance because training a GNN model from scratch to address previously unexplored vulnerability patterns across multiple domains, especially when data is scarce, can be prohibitively expensive and challenging to maintain.

Accordingly, we evaluated the effectiveness of the proposed model using four prominent GNN architectures: GGNN, GAT, GCN, and GraphSAGE. GAT with Transfer Learning outperformed all other models in the framework, achieving 99.06% accuracy, 98.50% precision, 100% recall, and an F1-score of 99.25%. This exceptional performance can be attributed to TL-GAT's ability to extract insights from a pre-

trained source domain, enhancing its overall predictive capacity.

While the vulnerability analysis workflow employed in this research is specifically designed for energy smart contracts and reentrancy vulnerabilities, it has the potential to be extended to other application domains or security defects when labeled contracts related to the desired vulnerability are available within the target domain.

# Chapter 5

# Conclusion and Future Work

## 5.1 Conclusion

Blockchain technology has brought innovation to a wide array of industries. In recent years, much attention has been given to blockchain platforms supporting smart contracts for the development of decentralized applications. Smart contracts provide the necessary versatility to consolidate diverse processes according to the requirements of the application. As a result, reliability of smart contracts is critical to the sustained growth of blockchain technology.

In this thesis, we undertook a venture aimed at enhancing vulnerability detection in smart contracts to increase the likelihood of their dependable and secure functioning. In light of recent advancements in the exploration of smart contract vulnerabilities, we introduced a domain-specific approach to enhance the accuracy of reentrancy detection.

In Chapter 2, we delve into the classification of smart contracts, setting the groundwork for a more profound understanding of violation structures and behavioral patterns of smart contracts across different application domains. as anticipated, upon the successful classification of smart contracts according to their underlying business logic, our findings confirm that there is no universally ideal contract analysis tool for all types of contracts and their respective business logic. Additionally, despite the evident association between smart contract categories and their vulnerabilities, it can be noted

that vulnerability analysis tools do not cover the wide range of violation structures and behavioral patterns present across different application domains. In order to address this gap, Chapter 3 of this thesis undertakes an assessment of benchmark vulnerability analysis tools using categorized and carefully curated contracts. Our examination led to the determination that energy contracts display security vulnerabilities that surpass the average, and their longer processing times contribute to a heightened risk of failures or timeouts. Moreover, it became evident that the accuracy of the detection tools fluctuates across domains, as the benchmark tools did not live up to the claimed level of precision in the curated contracts. This progression led us to the final phase of our research, Chapter 4, where we present a vulnerability detection framework grounded in the transfer learning within graph neural networks. This framework provides the capability to independently assess the vulnerability source in each domain, empowering developers to differentiate between domain-specific vulnerabilities, particularly when these contracts are utilized in diverse execution environments.

Below, we present a brief overview of the methodologies employed, primary outcomes, and significant contributions stemming from the presented research studies.

- **In Chapter 2, we introduce a framework that utilizes NLP in conjunction with machine learning classification algorithms to identify and accurately categorize energy smart contracts.** Detected contracts are further examined to discern any discrepancies or patterns in the distribution of code segments, the predominant use of specific functions, and recurring contracts across the Ethereum network. The proposed approach will help to establish the groundwork for innovative solutions for domain-specific classification and vulnerability detection of smart contracts. This approach enables the exclusive examination of the grammatical, symbolic, and arithmetic features of smart contracts, enhancing the detection of vulnerabilities that may have gone undetected.

- **In Chapter 3, we perform an analysis demonstrating the variability in the accuracy of vulnerability detection tools across different domains.** Vulnerability assessment is conducted on both energy-related and non-energy categories using benchmark analysis tools. The analysis underscores the substantial differences in the effects of specific vulnerabilities across various domains. This discrepancy highlights the need to prioritize vulnerability mitigation in alignment with the unique business logic of each domain. Examination of the code embedding of energy smart contracts and the assessment of their vulnerabilities within transactive energy systems further brings to light the revelation that energy contracts exhibit a notable number of defects. The findings confirm the need for more targeted and accurate vulnerability detection, encompassing both existing and emerging vulnerabilities from a domain-specific perspective. This analysis is a significant advancement in reinforcing the importance of dedicating more resources to tailor vulnerability detection tools to cater to the distinctive attributes of diverse application domains.

- **In Chapter 4, we suggest a framework that harnesses transfer learning with graph neural networks to enhance reentrancy detection in smart contracts**. We highlight the inefficiency and lack of adaptability in approaches that rely on expert-defined patterns. The proposed framework provides the capability to independently assess the vulnerability source in each domain, empowering developers to differentiate between domain-specific vulnerabilities, particularly when these contracts are utilized in diverse execution environments. The premise behind TL-GAT is to use transfer learning in order to repurpose a pre-trained GAT model for application to a new domain. Transfer learning allows for the use of already established knowledge about vulnerabilities to improve the generalisation of emerging vulnerabilities, with limited data.

## 5.2 Future Work

Although our thesis delves deeply into numerous strategies for enhancing the effectiveness of smart contract classification and vulnerability detection methods, offering practical solutions and enhancements, there are numerous opportunities for future research extensions. We enumerate a few of these potential research directions below:

- **Expansion to Other Domains**

  Extend the research to other domains, such as finance, healthcare, or supply chain management. Each domain has its own vocabulary, specifications, and regulatory concerns, making classification and vulnerability detection interesting areas. Exploring these diverse areas enables researchers to develop more targeted and specialised models for classification and vulnerability detection, to ensure the solutions are not just accurate but also highly relevant to the challenges inherent in these industries. This expansion broadens the potential applications and impact of the research, as it can address serious challenges and improve security and reliability within each domain.

- **Expansion to Other Vulnerabilities**

  Training models on a variety of data sources facilitates the recognition of patterns and vulnerabilities unique to each domain. By emphasising on these research directions, the field of smart contract vulnerability detection evolves to be more versatile, and aligned with the varied requirements of various industries, hence improving the security of decentralised applications. Training models across various domains provides to insights that are transferable between industries. Knowledge extracted from one domain allows to broaden our understanding of smart contract vulnerabilities while further developing detection abilities. Smart contract vulnerability detection's ultimate goal is to improve the security of decentralised applications. Domain-specific training allows for the development of targeted models that are advanced at flagging vulnerabilities and reducing

the risk of security breaches, ultimately protecting users and assets residing on the ledgers.

- **Scaling to Large Smart Contract Ecosystems**

  Blockchain networks are growing in scale and complexity, with a plethora of decentralised applications and smart contracts. This complexity brings with it a higher risk of vulnerabilities. Hence, scalable vulnerability detection methods are critical for keeping up with these ecosystems rapid development. This can be supported by investigating scalable methods for applying transfer learning to large ecosystems of smart contracts. As blockchain networks expand, scalable vulnerability detection becomes increasingly important for ensuring security. Many industries are governed by specific regulations, and scalable methods can aid in ensuring that smart contracts conform to these requirements efficiently.

# Bibliography


[1] A. Adeyemi *et al.*, "Blockchain technology applications in power distribution systems," *Electric Power Systems Research*, vol. 183, p. 106 817, 2020.

[2] R. Agarwal, T. Thapliyal, and S. Shukla, "Analyzing malicious activities and detecting adversarial behavior in cryptocurrency based permissionless blockchains: An ethereum use case," *Distrib. Ledger Technol. Res. Pract.*, vol. 1, pp. 1–21, 2022.

[3] N. Aidee, M. Johar, M. Alkawaz, A. Hajamydeen, and M. Al-Tamimi, "Vulnerability assessment on ethereum-based smart contract applications," in *Proceedings of the 2021 IEEE International Conference on Automatic Control and Intelligent Systems*, 2021, pp. 13–18.

[4] B. Altınel, M. Ganiz, and B. Diri, "A corpus-based semantic kernel for text classification by using meaning values of terms," *Eng. Appl. Artif. Intell.*, vol. 43, pp. 54–66, 2015.

[5] M. Andoni *et al.*, "Blockchain technology in the energy sector: A systematic review of challenges and opportunities," *Renewable and sustainable energy reviews*, vol. 100, pp. 143–174, 2019.

[6] N. Ashizawa, N. Yanai, J. P. Cruz, and S. Okamura, "Eth2vec: Learning contract-wide code representations for vulnerability detection on ethereum smart contracts," in *Proceedings of the 3rd ACM international symposium on blockchain and secure critical infrastructure*, 2021, pp. 47–59.

[7] S. Barj, A. Ouaddah, and A. Mezrioui, "Cryptography in distributed ledger technologies from a layered perspective: A state of the art," in *International Conference on Digital Technologies and Applications*, Springer, 2023, pp. 210–220.

[8] C. Barreto, T. Eghtesad, S. Eisele, A. Laszka, A. Dubey, and X. Koutsoukos, "Cyber-attacks and mitigation in blockchain-based transactive energy systems," in *Proceedings of the 2020 IEEE Conference on Industrial Cyberphysical Systems*, vol. 1, 2020, pp. 129–136.

[9] M. Bartoletti and L. Pompianu, "An empirical analysis of smart contracts: Platforms, applications, and design patterns," in *Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21*, Springer, 2017, pp. 494–509.

[10] N. Benisi, M. Aminian, and B. Javadi, "Blockchain-based decentralized storage networks: A survey," *Journal of Network and Computer Applications*, vol. 162, p. 102 656, 2020.

[11] V. D. Blondel, J. L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *J. Stat. Mech.-Theory Exp.*, vol. 2008, P10008, 2008.

[12] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," White Pap., Tech. Rep. 3, 2014, pp. 2–1.

[13] J. Cai, B. Li, J. Zhang, X. Sun, and B. Chen, "Combine sliced joint graph with graph neural networks for smart contract vulnerability detection," *Journal of Systems and Software*, vol. 195, p. 111 550, 2023.

[14] *CES Energy Solutions*, https://www.cesenergysolutions.com/, Accessed on 17 April 2023.

[15] J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo, and T. Chen, "Defining smart contract defects on ethereum," *IEEE Trans. Softw. Eng.*, vol. 48, pp. 327–345, 2020.

[16] Q. Chen, T. T. Wu, and M. Fang, "Detecting local community structure in complex networks based on local degree central nodes," *Physica A.*, vol. 392, pp. 529–537, 2013.

[17] Z. Chen, S. Kommrusch, and M. Monperrus, "Neural transfer learning for repairing security vulnerabilities in c code," *IEEE Transactions on Software Engineering*, vol. 49, no. 1, pp. 147–165, 2022.

[18] V. Chia *et al.*, "Rethinking blockchain security: Position paper," in *Proceedings of the 2018 IEEE International Conference on Internet of Things*, 2018, pp. 1273–1280.

[19] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Phys. Rev. E.*, vol. 70, p. 066 111, 2004.

[20] *Coinbase*, https://www.coinbase.com/price/solar-energy, Accessed on 17 April 2023.

[21] ConsenSys, *Consensys/surya: A set of utilities for exploring solidity contracts*, Accessed on September 27, 2023. [Online]. Available: https://github.com/ConsenSys/surya.

[22] *Creative energy*, https://creative.energy/, Accessed on 17 April 2023.

[23] U. Damisa, N. I. Nwulu, and P. Siano, "Towards blockchain-based energy trading: A smart contract implementation of energy double auction and spinning reserve trading," *Energies*, vol. 15, no. 11, p. 4084, 2022.

[24] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas, "Comparing community structure identification," *J. Stat. Mech.-Theory Exp.*, P09008, 2005.

[25] *Dapps - ethereum.* [Online]. Available: https://ethereum.org/en/dapps/.

[26] M. Demir, M. Alalfi, O. Turetken, and A. Ferworn, "Security smells in smart contracts," in *Proceedings of the 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion*, 2019, pp. 442–449.

[27] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv*, 2018. eprint: 1810.04805.

[28] T. Durieux, J. Ferreira, R. Abreu, and P. Cruz, "Empirical review of automated analysis tools on 47,587 ethereum smart contracts," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 530–541.

[29] S. Eisele *et al.*, "Blockchains for transactive energy systems: Opportunities, challenges, and approaches," *Computer*, vol. 53, pp. 66–76, 2020.

[30] *Electrify asia*, accessed on 17 April 2023. [Online]. Available: https://electrify.asia/.

[31] *Etherscan - ethereum block explorer*. [Online]. Available: https://etherscan.io/.

[32] D. R. Fabio, D. Fabio, and P. Carlo, "Profiling core-periphery network structure by random walkers," *Sci. Rep.*, vol. 3, p. 1467, 2013.

[33] B. Fabricio and Z. Liang, "Fuzzy community structure detection by particle competition and cooperation," *Soft Comput.*, vol. 17, pp. 659–673, 2013.

[34] N. Fadhel, F. Lombardi, L. Aniello, A. Margheri, and V. Sassone, "Towards a semantic modeling for threat analysis of iot applications: A case study on transactive energy," in *Proceedings of the Living in the Internet of Things (IoT 2019)*, London, UK, 2019.

[35] K. Fan, Z. Bao, M. Liu, A. V. Vasilakos, and W. Shi, "Dredas: Decentralized, reliable and efficient remote outsourced data auditing scheme with blockchain smart contract for industrial iot," *Future Generation Computer Systems*, vol. 110, pp. 665–674, 2020.

[36] J. Feist, G. Grieco, and A. Groce, "Slither: A static analysis framework for smart contracts," in *Proceedings of the 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, Montreal, QC, Canada, 2019, pp. 8–15.

[37] T. Ford, "Benchmarking ethereum smart contract static analysis tools," Ph.D. dissertation, Texas A&M University, College Station, TX, USA, 2022.

[38] S. Fortunato, "Community detection in graphs," *Phys. Rep.-Rev. Sec. Phys. Lett.*, vol. 486, pp. 75–174, 2010.

[39] S. Fortunato and M. Barthelemy, "Resolution limit in community detection," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 104, pp. 36–41, 2007.

[40] K. Gai, Y. Wu, L. Zhu, M. Qiu, and M. Shen, "Privacy-preserving energy trading using consortium blockchain in smart grid," *IEEE Trans. Ind. Inform.*, vol. 15, pp. 3548–3558, 2019.

[41] S. García-Méndez, M. Fernandez-Gavilanes, J. Juncal-Martínez, F. González-Castaño, and Ó. Seara, "Identifying banking transaction descriptions via support vector machine short-text classification based on a specialized labeled corpus," *IEEE Access*, vol. 8, pp. 61 642–61 655, 2020.

[42] A. Ghaleb and K. 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*, Virtual, 2020, pp. 415–427.

[43] *Gnrg*, accessed on 17 April 2023. [Online]. Available: https://gnrg.co/.

[44] T. Górski, "Reconfigurable smart contracts for renewable energy exchange with re-use of verification rules," *Appl. Sci.*, vol. 12, p. 5339, 2022.

[45] C. Goutte and E. Gaussier, "A probabilistic interpretation of precision, recall and f-score, with implication for evaluation," pp. 345–359, 2005.

[46] S. Gregory, "Fuzzy overlapping communities in networks," *J. Stat. Mech.-Theory Exp.*, P02017, 2011.

[47] R. Gupta, M. Patel, A. Shukla, and S. Tanwar, "Deep learning-based malicious smart contract detection scheme for internet of things environment," *Comput. Electr. Eng.*, vol. 97, p. 107 583, 2022.

[48] M. Hamilton, "Blockchain distributed ledger technology: An introduction and focus on smart contracts," *Journal of Corporate Accounting & Finance*, vol. 31, no. 2, pp. 7–12, 2020.

[49] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[50] D. Han, C. Zhang, J. Ping, and Z. Yan, "Smart contract architecture for decentralized energy trading and management based on blockchains," *Energy*, vol. 199, p. 117 417, 2020.

[51] J. Hartmann, J. Huppertz, C. Schamp, and M. Heitmann, "Comparing automated text classification methods," *Int. J. Res. Mark.*, vol. 36, pp. 20–38, 2019.

[52] D. He, R. Wu, X. Li, S. Chan, and M. Guizani, "Detection of vulnerabilities of blockchain smart contracts," *IEEE Internet Things J.*, vol. 10, pp. 12 178–12 185, 2023.

[53] Y. He, H. Dong, H. Wu, and Q. Duan, "Formal analysis of reentrancy vulnerabilities in smart contract based on cpn," *Electronics*, vol. 12, no. 10, p. 2152, 2023.

[54] P. Hegedűs, "Towards analyzing the complexity landscape of solidity based ethereum smart contracts," *Technologies*, vol. 7, no. 1, p. 6, 2019.

[55] M. Hossain, S. Sarkar, and M. Rahman, "Different machine learning based approaches of baseline and deep learning models for bengali news categorization," *International Journal of Computer Applications*, vol. 975, p. 8887, 2020.

[56] A. Hrga, T. Capuder, and I. P. Žarko, "Demystifying distributed ledger technologies: Limits, challenges, and potentials in the energy sector," *IEEE Access*, vol. 8, pp. 126 149–126 163, 2020.

[57] T. Hu *et al.*, "Transaction-based classification and detection approach for ethereum smart contract," *Inf. Process. Manag.*, vol. 58, p. 102 462, 2021.

[58] Q. Huang *et al.*, "A review of transactive energy systems: Concept and implementation," *Energy Rep.*, vol. 7, pp. 7804–7824, 2021.

[59] E. Hullermeier and M. Rifqi, "A fuzzy variant of the rand index for comparing clustering structures," in *in Proc. IFSA/EUSFLAT Conf.*, 2009, pp. 1294–1298.

[60] G. Ibba and M. Ortu, "Analysis of the relationship between smart contracts' categories and vulnerabilities," in *Proceedings of the 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering*, 2022, pp. 1212–1218.

[61] D. Jatnika, M. Bijaksana, and A. Suryani, "Word2vec model analysis for semantic similarities in english words," *Procedia Computer Science*, vol. 157, pp. 160–167, 2019.

[62] B. Jiang, Y. Liu, and W. K. Chan, "Contractfuzzer: Fuzzing smart contracts for vulnerability detection," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 259–269.

[63] P. Jiang, F. Guo, K. Liang, J. Lai, and Q. Wen, "Searchain: Blockchain-based private keyword search in decentralized storage," *Future Generation Computer Systems*, vol. 107, pp. 781–792, 2020.

[64] J. Jiao, S. Kan, S.-W. Lin, D. Sanan, Y. Liu, and J. Sun, "Semantic understanding of smart contracts: Executable operational semantics of solidity," in *2020 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2020, pp. 1695–1712.

[65] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "Zeus: Analyzing safety of smart contracts," in *Proceedings of the 25th Annual Network and Distributed System Security Symposium*, 2018.

[66] N. Kannengiesser, S. Lins, C. Sander, K. Winter, H. Frey, and A. Sunyaev, "Challenges and common solutions in smart contract development," *IEEE Trans. Softw. Eng.*, vol. 48, pp. 4291–4318, 2021.

[67] G. Kaur, A. Habibi Lashkari, I. Sharafaldin, and Z. Habibi Lashkari, "Smart contracts and defi security and threats," in *Understanding Cybersecurity Management in Decentralized Finance: Challenges, Strategies, and Trends*. Berlin/Heidelberg, Germany: Springer, 2023, pp. 91–111.

[68] V. Y. Kemmoe, W. Stone, J. Kim, D. Kim, and J. Son, "Recent advances in smart contracts: A technical overview and state of the art," *IEEE Access*, vol. 8, pp. 117 782–117 801, 2020.

[69] D. Kirli *et al.*, "Smart contracts in energy systems: A systematic review of fundamental approaches and implementations," *Renew. Sustain. Energy Rev.*, vol. 158, p. 112 013, 2022.

[70] N. Kshetri, "Can blockchain strengthen the internet of things?" *IT professional*, vol. 19, no. 4, pp. 68–72, 2017.

[71] S. Kushwaha, S. Joshi, D. Singh, M. Kaur, and H. Lee, "Ethereum smart contract analysis tools: A systematic review," *IEEE Access*, vol. 10, pp. 57 037–57 062, 2022.

[72] S. Kushwaha, S. Joshi, D. Singh, M. Kaur, and H. Lee, "Systematic review of security vulnerabilities in ethereum blockchain smart contracts," *IEEE Access*, vol. 10, pp. 6605–6621, 2022.

[73] M. Labani, P. Moradi, F. Ahmadizar, and M. Jalili, "A novel multivariate filter method for feature selection in text classification problems," *Engineering Applications of Artificial Intelligence*, vol. 70, pp. 25–37, 2018.

[74] A. Lancichinetti and S. Fortunato, "Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities," *Phys. Rev. E.*, vol. 80, p. 016 118, 2009.

[75] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Phys. Rev. E.*, vol. 78, p. 046 110, 2008.

[76] B. Lashkari and P. Musilek, "Detection and analysis of ethereum energy smart contracts," *Appl. Sci.*, vol. 13, no. 13, p. 6027, 2023.

[77] B. Lashkari and P. Musilek, "Evaluation of smart contract vulnerability analysis tools: A domain-specific perspective," *Information*, vol. 14, no. 10, p. 533, 2023.

[78] C. I. Law. "Code is law." Accessed on 11 November 2022. (2022), [Online]. Available: https://ethereumclassic.org/why-classic/code-is-law.

[79] J. Li, X. Wang, and J. Eustace, "Detecting overlapping communities by seed community in weighted complex networks," *Physica A.*, vol. 392, pp. 6125–6134, 2013.

[80] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[81] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE transactions on neural networks and learning systems*, 2021.

[82] J. Liu, "Fuzzy modularity and fuzzy community structure in networks," *Eur. Phys. J. B.*, vol. 77, pp. 547–557, 2010.

[83] W. Liu, M. Pellegrini, and X. Wang, "Detecting communities based on network topology," *Sci. Rep.*, vol. 4, p. 5739, 2014.

[84] Z. Liu *et al.*, "Rethinking smart contract fuzzing: Fuzzing with invocation ordering and important branch revisiting," *arXiv preprint arXiv:2301.03943*, 2023.

[85] H. Lou, S. Li, and Y. Zhao, "Detecting community structure using label propagation with weighted coherent neighborhood propinquity," *Physica A.*, vol. 392, pp. 3095–3105, 2013.

[86] L. Luu, D. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, Vienna, Austria, 2016, pp. 254–269.

[87] C. Ma, S. Liu, and G. Xu, "Hgat: Smart contract vulnerability detection method based on hierarchical graph attention network," *Journal of Cloud Computing*, vol. 12, no. 1, pp. 1–13, 2023.

[88] L. Marchesi, M. Marchesi, G. Destefanis, G. Barabino, and D. Tigano, "Design patterns for gas optimization in ethereum," in *Proceedings of the 2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, London, ON, Canada: IEEE, 2020, pp. 9–15.

[89] M. I. Mehar *et al.*, "Understanding a revolutionary and flawed grand experiment in blockchain: The dao attack," *Journal of Cases on Information Technology (JCIT)*, vol. 21, no. 1, pp. 19–32, 2019.

[90] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao, "Deep learning–based text classification: A comprehensive review," *ACM Comput. Surv.*, vol. 54, pp. 1–40, 2021.

[91] M. Mironczuk and J. Protasiewicz, "A recent overview of the state-of-the-art elements of text classification," *Expert Syst. Appl.*, vol. 106, pp. 36–54, 2018.

[92] A. Monrat, O. Schelén, and K. Andersson, "A survey of blockchain from the perspectives of applications, challenges, and opportunities," *IEEE Access*, vol. 7, pp. 117 134–117 151, 2019.

[93] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized business review*, 2008.

[94] T. Nepusz, A. Petróczi, L. Négyessy, and F. Bazsó, "Fuzzy communities and the concept of bridgeness in complex networks," *Phys. Rev. E.*, vol. 77, p. 016 107, 2008.

[95] W. Network, *Wepower network*, Accessed on 12 May 2023. [Online]. Available: https://www.blockdata.tech/profiles/wepower.

[96] M. E. J. Newman, *Network data*, http://www-personal.umich.edu/~mejn/netdata/, 2013.

[97] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Phys. Rev. E.*, vol. 69, p. 026 113, 2004.

[98] I. Nikolić, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, "Finding the greedy, prodigal, and suicidal contracts at scale," in *Proceedings of the 34th Annual Computer Security Applications Conference*, 2018, pp. 653–663.

[99] A. Norta and T. Lepikult, "An evaluation framework for smart contract vulnerability detection tools on the ethereum blockchain,"

[100] M. Nour, J. P. Chaves-Ávila, and Á. Sánchez-Miralles, "Review of blockchain potential applications in the electricity sector and challenges for large scale adoption," *IEEE Access*, vol. 10, pp. 47 384–47 418, 2022.

[101] G. Oliva, A. Hassan, and Z. Jiang, "An exploratory study of smart contracts in the ethereum blockchain platform," *Empirical Software Engineering*, vol. 25, pp. 1864–1904, 2020.

[102] A. Pinna, S. Ibba, G. Baralla, R. Tonelli, and M. Marchesi, "A massive analysis of ethereum smart contracts: Empirical study and code metrics," *IEEE Access*, vol. 7, pp. 78 194–78 213, 2019.

[103] *Powerledger*, accessed on 12 May 2023. [Online]. Available: https://www.powerledger.io/.

[104] I. Psorakis, S. Roberts, M. Ebden, and B. Sheldon, "Overlapping community detection using bayesian non-negative matrix factorization," *Phys. Rev. E.*, vol. 83, p. 066 114, 2011.

[105] S. Qaiser and R. Ali, "Text mining: Use of tf-idf to examine the relevance of words to documents," *International Journal of Computer Applications*, vol. 181, pp. 25–29, 2018.

[106] P. Qian, Z. Liu, Q. He, R. Zimmermann, and X. Wang, "Towards automated reentrancy detection for smart contracts based on sequential models," *IEEE Access*, vol. 8, pp. 19 685–19 695, 2020.

[107] S. Qian, H. Ning, Y. He, and M. Chen, "Multi-label vulnerability detection of smart contracts based on bi-lstm and attention mechanism," *Electronics*, vol. 11, no. 19, p. 3260, 2022.

[108] U. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Phys. Rev E.*, vol. 76, p. 036 106, 2007.

[109] H. Rameder, M. Di Angelo, and G. Salzer, "Review of automated vulnerability analysis of smart contracts on ethereum," *Front. Blockchain*, vol. 5, p. 814 977, 2022.

[110] T. Roth, M. Utz, F. Baumgarte, A. Rieger, J. Sedlmeir, and J. Strüker, "Electricity powered by blockchain: A review with a european perspective," *Applied Energy*, vol. 325, p. 119 799, 2022.

[111] S. Sayeed, H. Marco-Gisbert, and T. Caira, "Smart contract: Attacks and protections," *IEEE Access*, vol. 8, pp. 24 416–24 427, 2020.

[112] D. Sebastian-Cardenas *et al.*, "Cybersecurity and privacy aspects of smart contracts in the energy domain," in *Proceedings of the 2022 IEEE 1st Global Emerging Technology Blockchain Forum: Blockchain & Beyond (iGETblockchain)*, Irvine, CA, USA, 2022, pp. 1–6.

[113] C. Sendner *et al.*, "Smarter contracts: Detecting vulnerabilities in smart contracts with deep transfer learning.," in *NDSS*, 2023.

[114] M. Setyawan, R. Awangga, and S. Efendi, "Comparison of multinomial naive bayes algorithm and logistic regression for intent classification in chatbot," in *2018 International Conference on Applied Engineering (ICAE)*, 2018, pp. 1–5.

[115] S. Seven, G. Yao, A. Soran, A. Onen, and S. Muyeen, "Peer-to-peer energy trading in virtual power plant based on blockchain smart contracts," *IEEE Access*, vol. 8, pp. 175 713–175 726, 2020.

[116] K. Shah, H. Patel, D. Sanghvi, and M. Shah, "A comparative analysis of logistic regression, random forest and knn models for text classification," *Augmented Human Research*, vol. 5, pp. 1–16, 2020.

[117] C. Shi, Y. Xiang, R. Doss, J. Yu, K. Sood, and L. Gao, "A bytecode-based approach for smart contract classification," *arXiv*, 2021, arXiv:2106.15497.

[118] C. Shi, Y. Xiang, J. Yu, L. Gao, K. Sood, and R. Doss, "A bytecode-based approach for smart contract classification," in *Proceedings of the 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering*, 2022, pp. 1046–1054.

[119] E. Sifra, "Security vulnerabilities and countermeasures of smart contracts: A survey," in *Proceedings of the 2022 IEEE International Conference on Blockchain (Blockchain)*, Espoo, Finland, 2022, pp. 512–515.

[120] A. Singh, R. Parizi, Q. Zhang, K. Choo, and A. Dehghantanha, "Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities," *Comput. Secur.*, vol. 88, p. 101 654, 2020.

[121] Smartbugs. "Smartbugs/smartbugs: Smartbugs: A framework to analyze ethereum smart contracts." Accessed on 27 September 2023. (2023), [Online]. Available: https://github.com/smartbugs/smartbugs.

[122] S. Sobolevsky and R. Campari, "General optimization technique for high-quality community detection in complex networks," *Phys. Rev. E.*, vol. 90, p. 012 811, 2014.

[123] Solhint. "Solhint." Accessed on 27 September 2023. (2023), [Online]. Available: https://protofire.github.io/solhint/.

[124] Solidity. "Solidity by example - call." Accessed on September 27, 2023. (), [Online]. Available: https://solidity-by-example.org/call/.

[125] Solidity. "Solidity by example - sending ether." Accessed on September 27, 2023. (), [Online]. Available: https://solidity-by-example.org/sending-ether/.

[126] P. Sun, L. Gao, and S. Han, "Identification of overlapping and non-overlapping community structure by fuzzy clustering in complex networks," *Inf. Sci.*, vol. 181, pp. 1060–1071, 2011.

[127] X. Sun, X. Lin, and Z. Liao, "An abi-based classification approach for ethereum smart contracts," in *2021 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, 2021, pp. 99–104.

[128] *Sunchain*, accessed on 17 April 2023. [Online]. Available: https://www.sunchain.fr/.

[129] S. Suneja, Y. Zheng, Y. Zhuang, J. Laredo, and A. Morari, "Learning to map source code to software vulnerability using code-as-a-graph," *arXiv preprint arXiv:2006.08614*, 2020.

[130] N. Szabo, "Smart contracts: Building blocks for digital markets," *Extropy Journal of Transhuman Thought*, vol. 18, pp. 50–53, 1996.

[131] N. Szabo, "Formalizing and securing relationships on public networks," *First monday*, 1997.

[132] H. T., B. J., L. C. R. K., and P. M., "A soft modularity function for detecting fuzzy communities in social networks," *IEEE Trans. Fuzzy Syst.*, vol. 21, pp. 1170–1175, 2013.

[133] W. J.-W. Tann, X. J. Han, S. S. Gupta, and Y.-S. Ong, "Towards safer smart contracts: A sequence learning approach to detecting security threats," *arXiv preprint arXiv:1811.06632*, 2018.

[134] I. Tenney, D. Das, and E. Pavlick, "Bert rediscovers the classical nlp pipeline," *arXiv*, 2019. eprint: 1905.05950.

[135] G. Tian, Q. Wang, Y. Zhao, L. Guo, Z. Sun, and L. Lv, "Smart contract classification with a bi-lstm based approach," *IEEE Access*, vol. 8, pp. 43 806–43 816, 2020.

[136] G. Tian, Q. Wang, Y. Zhao, L. Guo, Z. Sun, and L. Lv, "Smart contract classification with a bi-lstm based approach," *IEEE Access*, vol. 8, pp. 43 806–43 816, 2020. DOI: 10.1109/ACCESS.2020.2977362.

[137] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "Smartcheck: Static analysis of ethereum smart contracts," pp. 9–16, 2018.

[138] L. Toderean *et al.*, "A lockable erc20 token for peer to peer energy trading," *arXiv*, 2021, arXiv:2111.04467.

[139] P. Tolmach, Y. Li, S. Lin, Y. Liu, and Z. Li, "A survey of smart contract formal specification and verification," *ACM Computing Surveys*, vol. 54, pp. 1–38, 2021.

[140] C. Torres, A. Iannillo, A. Gervais, and R. State, "Confuzzius: A data dependency-aware hybrid fuzzer for smart contracts," pp. 103–119, 2021.

[141] C. Torres, J. Schütte, and R. State, "Osiris: Hunting for integer bugs in ethereum smart contracts," in *Proceedings of the 34th Annual Computer Security Applications Conference*, 2018, pp. 664–676.

[142] T. A. Usman, A. A. Selçuk, and S. Özarslan, "An analysis of ethereum smart contract vulnerabilities," in *2021 International Conference on Information Security and Cryptology (ISCTURKEY)*, IEEE, 2021, pp. 99–104.

[143] A. Uysal and S. Gunal, "The impact of preprocessing on text classification," *Inf. Process. Manag.*, vol. 50, pp. 104–112, 2014.

[144] A. Vacca, A. Di Sorbo, C. A. Visaggio, and G. Canfora, "A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges," *Journal of Systems and Software*, vol. 174, p. 110 891, 2021.

[145] C. Vehlow, T. Reinhardt, and D. Weiskopf, "Visualizing fuzzy overlapping communities in networks," *IEEE Trans. Vis. Comput. Graph.*, vol. 19, pp. 2486–2495, 2013.

[146] F. Victor and B. Lüders, "Measuring ethereum-based erc20 token networks," in *Proceedings of the International Conference on Financial Cryptography and Data Security*, Springer, 2019, pp. 113–129.

[147] G. Vieira and J. Zhang, "Peer-to-peer energy trading in a microgrid leveraged by smart contracts," *Renew. Sustain. Energy Rev.*, vol. 143, p. 110 900, 2021.

[148] L. Šubelj and M. Bajec, "Robust network community detection using balanced propagation," *Eur. Phys. J. B.*, vol. 81, pp. 353–362, 2011.

[149] L. Šubelj and M. Bajec, "Ubiquitousness of link-density and link-pattern communities in real-world networks," *Eur. Phys. J. B.*, vol. 85, pp. 1–11, 2012.

[150] L. Šubelj and M. Bajec, "Unfolding communities in large complex networks: Combining defensive and offensive label propagation for core extraction," *Phys. Rev. E.*, vol. 83, p. 036 103, 2011.

[151] D. Vujičić, D. Jagodić, and S. Ranđić, "Blockchain technology, bitcoin, and ethereum: A brief overview," in *2018 17th international symposium infoteh-jahorina (infoteh)*, IEEE, 2018, pp. 1–6.

[152] L. Wang, H. Cheng, Z. Zheng, A. Yang, and X. Zhu, "Ponzi scheme detection via oversampling-based long short-term memory for smart contracts," *Knowl.-Based Syst.*, vol. 228, p. 107 312, 2021.

[153] W. Wang, D. Liu, X. Liu, and L. Pan, "Fuzzy overlapping community detection based on local random walk and multidimensional scaling," *Physica A.*, vol. 392, pp. 6578–6586, 2013.

[154] W. Wang, J. Song, G. Xu, Y. Li, H. Wang, and C. Su, "Contractward: Automated vulnerability detection models for ethereum smart contracts," *IEEE Trans. Netw. Sci. Eng.*, vol. 8, pp. 1133–1144, 2020.

[155] X. Wang and J. Li, "Detecting communities by the core-vertex and intimate degree in complex networks," *Physica A.*, vol. 392, pp. 2555–2563, 2013.

[156] Y. Wang, G. Gou, C. Liu, M. Cui, Z. Li, and G. Xiong, "Survey of security supervision on blockchain from the perspective of technology," *Journal of Information Security and Applications*, vol. 60, p. 102 859, 2021.

[157] Z. Wang, W. Wu, C. Zeng, J. Yao, Y. Yang, and H. Xu, "Smart contract vulnerability detection for educational blockchain based on graph neural networks," in *2022 International Conference on Intelligent Education and Intelligent Research (IEIR)*, IEEE, 2022, pp. 8–14.

[158] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[159] *Wpp energy*, accessed on 17 April 2023. [Online]. Available: https://wppenergy.com/.

[160] C. Wu, J. Xiong, H. Xiong, Y. Zhao, and W. Yi, "A review on recent progress of smart contract in blockchain," *IEEE Access*, vol. 10, pp. 50 839–50 863, 2022.

[161] H. Wu *et al.*, "Peculiar: Smart contract vulnerability detection based on crucial data flow graph and pre-training techniques," in *Proceedings of the 2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, Wuhan, China, 2021, pp. 378–389.

[162] H. Wu, H. Dong, Y. He, and Q. Duan, "Smart contract vulnerability detection based on hybrid attention mechanism model," *Applied Sciences*, vol. 13, no. 2, p. 770, 2023.

[163] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[164] Y. Xue, M. Ma, Y. Lin, Y. Sui, J. Ye, and T. Peng, "Cross-contract static analysis for detecting practical reentrancy vulnerabilities in smart contracts," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, Melbourne, Australia, 2020, pp. 1029–1040.

[165] X. Yu, H. Zhao, B. Hou, Z. Ying, and B. Wu, "Deescvhunter: A deep learning-based framework for smart contract vulnerability detection," in *2021 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2021, pp. 1–8.

[166] F. Zablith and I. Osman, "Reviewmodus: Text classification and sentiment prediction of unstructured reviews using a hybrid combination of machine learning and evaluation models," *Appl. Math. Model.*, vol. 71, pp. 569–583, 2019.

[167] H. Zhang and G. Zhong, "Improving short text classification by learning vector representations of both words and hidden topics," *Knowledge-Based Systems*, vol. 102, pp. 76–86, 2016.

[168] L. Zhang *et al.*, "A novel smart contract reentrancy vulnerability detection model based on bigas," *Journal of Signal Processing Systems*, pp. 1–23, 2023.

[169] P. Zhang, F. Xiao, and X. Luo, "A framework and dataset for bugs in ethereum smart contracts," in *Proceedings of the 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Adelaide, Australia, 2020, pp. 139–150.

[170] S. Zhang, D. May, M. Gül, and P. Musilek, "Reinforcement learning-driven local transactive energy market for distributed energy resources," *Energy AI*, vol. 8, p. 100 150, 2022.

[171] S. Zhang, R. Wang, and X. Zhang, "Identification of overlapping community structure in complex networks using fuzzy c-means clustering," *Physica A.*, vol. 374, pp. 483–490, 2007.

[172] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: A comprehensive review," *Computational Social Networks*, vol. 6, no. 1, pp. 1–23, 2019.

[173] Y. Zhang and D. Yeung, "Overlapping community detection via bounded nonnegative matrix tri-factorization," in *In Proc. ACM SIGKDD Conf.*, 2012, pp. 606–614.

[174] Z. Zhang *et al.*, "Reentrancy vulnerability detection and localization: A deep learning based two-phase approach," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, Rochester, MI, USA, 2022, pp. 1–13.

[175] X. Zhao, Z. Chen, X. Chen, Y. Wang, and C. Tang, "The dao attack paradoxes in propositional logic," in *2017 4th international conference on systems and informatics (ICSAI)*, IEEE, 2017, pp. 1743–1746.

[176] Z. Zheng *et al.*, "An overview of smart contracts: Challenges, advances, and platforms," *Future Gener. Comput. Syst.*, vol. 105, pp. 475–491, 2020.

[177] J. Zhou *et al.*, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

[178] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, "Smart contract vulnerability detection using graph neural network," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, C. Bessiere, Ed., Main track, International Joint Conferences on Artificial Intelligence Organization, Jul. 2020, pp. 3283–3290. DOI: 10.24963/ijcai.2020/454. [Online]. Available: https://doi.org/10.24963/ijcai.2020/454.

[179] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, "Smart contract vulnerability detection using graph neural networks," in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 2021, pp. 3283–3290.