

# Validation Framework for E-Contract and Smart Contract

Sangharatna Godbole, P. Radha Krishna, Sunkara Sri Harika, and Pooja Varnam

NITMiner Technologies, Department of Computer Science and Engineering

National Institute of Technology Warangal,

Warangal, Telangana, India

{sanghu,prkrishna}@nitw.ac.in,{ss952022,vp842026}@student.nitw.ac.in.

## Abstract

We propose and develop a framework for validating smart contracts derived from e-contracts. The goal is to ensure the generated smart contracts fulfil all the conditions outlined in their corresponding e-contracts. By confirming alignment between the smart contracts and their original agreements, this approach enhances trust and reliability in automated contract execution. The proposed framework will systematically compare and validate the terms and clauses of the e-contracts with the logic of the smart contracts. This validation confirms that the agreement is accurately translated into executable code. Automated verification identifies issues between the e-contracts and their smart contract counterparts. This proposed work will solve the problems of gap between legal language and code execution, this framework ensures seamless integration of smart contracts into the existing legal framework.

## CCS Concepts

• **Software and its engineering** → **Software testing and debugging**.

## Keywords

E-contract, Smart contract, Knowledge graph, Abstract Syntax Tree, Solidity

## ACM Reference Format:

Sangharatna Godbole, P. Radha Krishna, Sunkara Sri Harika, and Pooja Varnam. 2018. Validation Framework for E-Contract and Smart Contract. In *Proceedings of The 29th International Conference on Evaluation and Assessment in Software Engineering (EASE 2025)*. ACM, New York, NY, USA, 6 pages. <https://doi.org/XXXXXXX.XXXXXXX>

## 1 Introduction

The rapid advancement of technology has led to a shift from traditional paper-based contracts to digital contracts, commonly known as e-contracts. These e-contracts provide the convenience of electronic storage, quick accessibility, and ease of sharing. However, with the rise of blockchain technology, a new type of contract has emerged Smart Contract. Smart contracts are self-executing agreements with the terms and conditions written directly in code. They offer automated enforcement, transparency and tamper-resistant

execution. While smart contracts offer significant advantages over traditional contracts, their development process raises important concerns [3–5].

Smart contracts are created based on the terms and conditions specified in natural language within e-contracts [11]. The created smart contract code may be syntactically and logically correct, but it lacks the essential conditions outlined in the e-contract. The deployed smart contract may not meet the end user's expectations and requirements. This leads to potential contract failures and unsatisfactory outcomes that can lead to financial losses and legal disputes. So, one of the challenges is ensuring that smart contracts can be created and carried out under the party's willingness.

We aim to ensure that smart contracts function correctly and comply with legal regulations, addressing concerns about their security and reliability. Our research compares e-contracts and smart contracts using advanced techniques like pre-processing, dependency parsing, and constructing knowledge graphs. We analyse important entities in both contract types and identify any differences or discrepancies.

For e-contracts, we refine the text and extract significant entities using pre-processing and dependency parsing. Next, we create a knowledge graph to get the relationships within the contract. Also, we generate knowledge graphs for smart contracts by generating Abstract Syntax Trees (AST) to extract information. By comparing the entities extracted from both types of contracts, we aim to highlight similarities, differences, and potential discrepancies. This research contributes to understanding how advanced techniques can improve the analysis of contracts, benefiting stakeholders in the legal and blockchain fields.

The rest of the paper is structured as follows: Section 2 deals with the Background, and Section 3 explains the Motivation. Section 4 shows the proposed approach with algorithms and frameworks. Section 5 discusses the experimental results. Finally, we conclude the paper and discuss future work in Section 6.

## 2 Background

Natural Language Processing (NLP) techniques [8, 9] have been increasingly utilised in legal and contractual domains to extract meaningful insights from textual data. It is used to perform the automated analysis of e-contracts, extracting important entities, clauses, and relationships embedded within the text. The techniques, such as pre-processing, dependency parsing, and entity recognition, are used to examine the linguistic structure of e-contracts to get valuable information.

Knowledge graph[1] generation plays a key role in enhancing the understanding of contractual agreements by representing entities and their relationships in a structured format. In e-contracts, knowledge graphs capture the semantics and connections between

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

EASE 2025, Istanbul, Türkiye

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-XXXX-X/2018/06

<https://doi.org/XXXXXXX.XXXXXXX>

contractual clauses, commitments, and parties involved. Creating a knowledge graph based on the extracted entities and their dependencies, we get the relationships within e-contracts. This helps for a deeper understanding of the contractual responsibilities and commitment which facilitates the identification of discrepancies or inconsistencies between different clauses or parties. Knowledge graph generation is a powerful tool for visualising and navigating the complex landscape of contractual agreements, ultimately validating and verifying e-contracts against their corresponding smart contracts.

Literature shows innovative techniques such as the Software Word Usage Model (SWUM)[2] in automatic comment generation for programming languages like Java. SWUM has shown promise in generating comments based on the usage of words within the code [12]. Applying a universal SWUM to languages like Solidity, specifically designed for blockchain operations, presents challenges. Solidity incorporates numerous reserved terms unique to blockchain, which may hinder the effectiveness of a generic SWUM.

### 3 Motivation

We have proposed and designed a validation system for smart contracts and e-contracts. This will ensure the smart contracts' integrity, security, and functionality and will verify that they accurately represent the terms and conditions outlined in the e-contracts.

We have identified some challenges in this domain. **Complexity Discrepancy:** Smart contracts, implemented in languages like Solidity, can have complexities which have not been fully captured in their e-contract representations. This leads to discrepancies in functionality and execution.

**Regulatory Compliance:** It ensures that smart contracts adhere to legal regulations outlined in e-contracts, posing a significant challenge, particularly in jurisdictions with evolving blockchain and smart contract laws.

**Semantic Alignment:** Variations in language and structure between e-contracts and smart contracts may hinder accurate semantic alignment, making it difficult to verify if all contractual obligations are faithfully translated into executable code.

### 4 Proposed Approach

Our research employs advanced pre-processing, dependency parsing, and knowledge graph construction to analyze and compare e-contracts and smart contracts. Our approach involves extracting and analyzing important entities from both contract types.

#### 4.1 Algorithms

The proposed methodology begins with the construction of a knowledge graph from the natural language e-contract, as detailed in Algorithm 1. The process starts by preprocessing the e-contract text to eliminate noise such as special characters and irrelevant symbols, followed by tokenization and normalization. Using Natural Language Processing (NLP) techniques, particularly dependency parsing, key entities (e.g., parties, obligations, and dates) and their semantic relationships are extracted. These entities serve as the nodes and the relationships as the edges in the resulting knowledge graph  $\mathcal{G}_e = (V_e, E_e)$ . This graph-based structure enables a formal

representation of the contract that is both interpretable and suitable for automated reasoning and visualization.

In parallel, a knowledge graph is also constructed from the smart contract, as described in Algorithm 2. The Solidity source code is analyzed by first extracting the compiler version and generating the Abstract Syntax Tree (AST) through compilation. The AST contains the hierarchical structure of the contract, including its functions, variables, and control logic. Using a custom grammar-driven approach, these elements are translated into human-readable descriptions that represent the functional semantics of the smart contract. Entities and dependencies derived from this semantic representation are used to build the smart contract knowledge graph  $\mathcal{G}_s = (V_s, E_s)$ . Finally, as shown in Algorithm 3, the two knowledge graphs  $\mathcal{G}_e$  and  $\mathcal{G}_s$  are compared by matching their nodes and edges to identify similarities and discrepancies. This comparison as shown in Algorithm 4, provides a structured validation framework to ensure that the smart contract faithfully implements the specifications outlined in the original e-contract.

---

#### Algorithm 1 E-Contract Knowledge Graph Construction

---

**Require:** Raw e-contract text  $T_e$   
**Ensure:** Knowledge graph  $\mathcal{G}_e = (V_e, E_e)$

- 1:  $T'_e \leftarrow \text{Preprocess}(T_e)$  {Remove noise, tokenize}
- 2:  $V_e \leftarrow \text{ExtractEntities}(T'_e)$
- 3:  $E_e \leftarrow \text{ExtractRelations}(T'_e)$
- 4:  $\mathcal{G}_e \leftarrow (V_e, E_e)$
- 5: **return**  $\mathcal{G}_e$

---



---

#### Algorithm 2 Smart Contract Knowledge Graph Construction

---

**Require:** Solidity source code  $S$   
**Ensure:** Knowledge graph  $\mathcal{G}_s = (V_s, E_s)$

- 1:  $v \leftarrow \text{ExtractSolidityVersion}(S)$
- 2:  $\text{Compiler} \leftarrow \text{SelectCompiler}(v)$
- 3:  $\text{AST} \leftarrow \text{CompileAndGenerateAST}(S, \text{Compiler})$
- 4:  $\text{AST}_{\text{json}} \leftarrow \text{SaveAsJSON}(\text{AST})$
- 5: **for** each node  $n \in \text{AST}$  **do**
- 6:    $t \leftarrow \text{NodeType}(n)$
- 7:    $d \leftarrow \text{ApplyGrammar}(t)$
- 8:   Add description  $d$  to semantic structure
- 9: **end for**
- 10:  $V_s \leftarrow \text{ExtractEntitiesFromSemanticStructure}()$
- 11:  $E_s \leftarrow \text{ExtractRelationsFromSemanticStructure}()$
- 12:  $\mathcal{G}_s \leftarrow (V_s, E_s)$
- 13: **return**  $\mathcal{G}_s$

---

#### 4.2 Knowledge Graph construction of E-Contract

Fig. 1 shows the framework for the generation of a knowledge graph from E-Contract. Utilizing Natural Language Processing (NLP) techniques and dependency parsing, we generated a knowledge graph from e-contracts [1]. This process involved extracting entities and

**Algorithm 3** Compare Knowledge Graphs**Require:**  $\mathcal{G}_e = (V_e, E_e)$ ,  $\mathcal{G}_s = (V_s, E_s)$ **Ensure:** Discrepancy report  $\Delta$ 

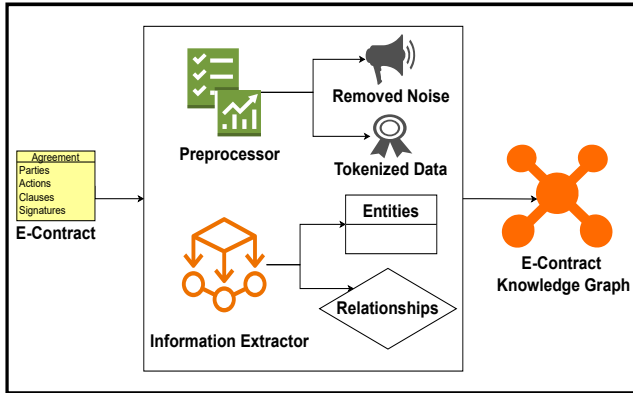
- 1:  $M_V \leftarrow \text{MatchEntities}(V_e, V_s)$
- 2:  $M_E \leftarrow \text{MatchRelations}(E_e, E_s)$
- 3:  $\Delta_V \leftarrow V_e \cup V_s \setminus M_V$
- 4:  $\Delta_E \leftarrow E_e \cup E_s \setminus M_E$
- 5:  $\Delta \leftarrow (\Delta_V, \Delta_E)$
- 6: **return**  $\Delta$

**Algorithm 4** Integrated Contract Analysis and Comparison**Require:** E-contract text  $T_e$ , smart contract code  $S$ **Ensure:** Validation report  $\Delta$ 

- 1:  $\mathcal{G}_e \leftarrow \text{E-CONTRACT KNOWLEDGE GRAPH CONSTRUCTION}(T_e)$   
    {Call Algorithm 1}
- 2:  $\mathcal{G}_s \leftarrow \text{SMART CONTRACT KNOWLEDGE GRAPH CONSTRUCTION}(S)$   
    {Call Algorithm 2}
- 3:  $\Delta \leftarrow \text{COMPARE KNOWLEDGE GRAPHS}(\mathcal{G}_e, \mathcal{G}_s)$  {Call Algorithm 3}
- 4: **return**  $\Delta$

identifying relationships between them embedded within the contract's textual data. We used the NetworkX library[6] to generate graphs and matplotlib<sup>1</sup> to visualise the graphs.

- Preprocess the text to remove or modify any unwanted characters or symbols.
- Extract entities and identify the relation between them.
- Nodes contain entities, and the edges represent the relation between the nodes.

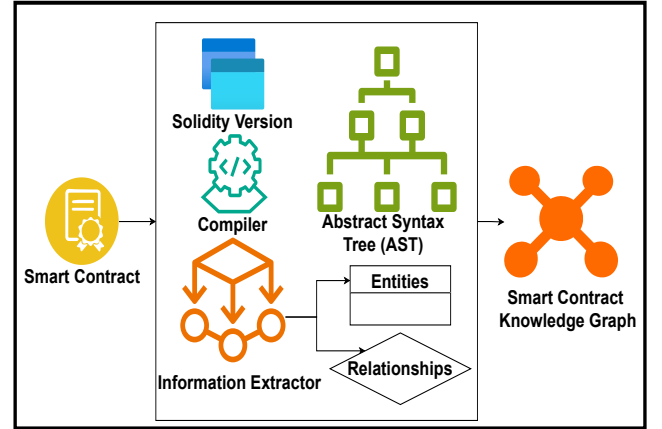


**Figure 1: Framework for generation of E-Contract Knowledge Graph**

### 4.3 Knowledge Graph Generation for Smart Contract

Fig. 2 shows the Knowledge Graph Generation for Smart Contract into the Knowledge Graph.

<sup>1</sup><https://matplotlib.org/>



**Figure 2: Framework for generation of Smart Contract Knowledge Graph**

Abstract Syntax Tree (AST) generation begins by reading the contents of the specified Solidity smart contract file [10]. It then extracts the Solidity version indicated in the contract's pragma statement, which is crucial for selecting the compatible Solidity compiler version. The extracted solidity version is utilized, and the program selects the appropriate Solidity compiler version to compile the smart contract and avoid compile time issues.

Following the compilation process, the program captures the compiler output, including any developer documentation generated during compilation. This documentation contains detailed insights into the contract's methods and functions, including their descriptions and parameters [3].

After parsing the developer documentation from the compiler output, the program generates the Solidity smart contract's Abstract Syntax Tree (AST). The AST represents the smart contract's structure and syntactic elements. It is saved in JSON format, which helps understand the smart contract's structure and supports code review and debugging. The program automatically creates the AST, which makes analyzing and validating smart contracts faster and more accurate.

In our approach, we parsed the generated Abstract Syntax Tree (AST) based on node types, utilizing this structured representation of the contract's syntax to facilitate the translation of code into human-readable language [10]. We have designed a grammar that is connected to the specific node types encountered in the AST. This decodes the smart contract's logic and functionality systematically. We developed a framework for interpreting the contract's syntax by classifying the nodes based on their types, such as variables, functions, conditionals, and loops.

For instance, variables and their attributes were mapped to grammar rules specifying their data types, names, and initialization values. The functions were translated into human-readable descriptions, which described their parameters, return types, and operations within the function body. The conditional statements and loops were parsed to capture their logical conditions and iterative behaviours. This shows clear explanations of control flow within the

smart contract. We were able to systematically traverse the AST by connecting to this grammar-driven approach. We extracted relevant information from each node and composed readable descriptions of the smart contract's structure and behaviour. The stakeholders understand its logic and implications in a human-readable format. Additionally, it enhanced the accessibility of smart contract code, enabling non-technical stakeholders to engage meaningfully with the contract's contents and implications.

```

1 Parties: This Smart Rental Contract (the "Contract") is
  entered into on 01/01/2025, between.
2
3 Landlord: ABC, residing in London, United Kingdom
4
5 Tenant: XYZ, residing at Singapore, Singapore
6
7 Property: The Landlord agrees to rent the following
  property to the Tenant:
8
9 Address: 123 Baker Street, London, NW1 4NW
10
11 Term: The term of this Contract shall be from 01/01/2025
  to 31/12/2025.
12
13 Rent: The Tenant agrees to pay a monthly rent of GBP
  5000. Rent is due on the first day of each month.
14
15 Security Deposit: The tenant shall deposit GBP 2000 as a
  security deposit upon signing this Contract. The
  security deposit will be held by the Landlord and
  returned to the Tenant upon termination of the
  Contract, subject to any deductions for damages or
  unpaid rent.
16
17 Utilities: The Tenant is responsible for paying all
  utility charges, including electricity, water, gas,
  and internet.
18
19 Use of Property: The Tenant shall use the property solely
  as a private residence and shall not engage in any
  unlawful activities on the premises.
20
21 Maintenance and Repairs: The Landlord is responsible for
  maintaining the property in good condition. The
  Tenant shall keep the property clean and sanitary.
22
23 Termination: Either party may terminate this Contract
  with one month written notice to the other party.
24
25 Governing Law: This Contract shall be governed by the
  laws of London, UK.
26
27 Signatures: In witness whereof, the parties hereto have
  executed this Contract on the date first above
  written.
28
29 [Landlord's Signature]: Date: 01/01/2025 ABC
30
31 [Tenant's Signature]: Date: 01/01/2025 XYZ

```

**Listing 1: E-Contract for Rental Agreement**

Our methodology used the parsed data, representing the smart contract in natural language, to construct a new knowledge graph. This knowledge graph encapsulates the semantic relationships and dependencies extracted from the human-readable representation of the contract's logic and functionality.

Once both knowledge graphs—derived from the original e-contract and the parsed smart contract—are constructed, we compare them systematically based on identifying similarities, differences, and potential discrepancies between the entities, relationships, and dependencies in the knowledge graphs.

```

1 pragma solidity ^0.8.0;
2 contract RentalAgreement {
3     address public landlord;
4     address public tenant;
5     string public propertyAddress;
6     uint256 public rentAmount;
7     uint256 public securityDeposit;
8     uint256 public termStartDate;
9     uint256 public termEndDate;
10    bool public terminated;
11    event RentPaid(uint256 amount);
12    event SecurityDepositPaid(uint256 amount);
13    event ContractTerminated(address terminatedBy);
14    constructor(
15        address _landlord,
16        address _tenant,
17        string memory _propertyAddress,
18        uint256 _rentAmount,
19        uint256 _securityDeposit,
20        uint256 _termStartDate,
21        uint256 _termEndDate
22    ) {
23        landlord = _landlord;
24        tenant = _tenant;
25        propertyAddress = _propertyAddress;
26        rentAmount = _rentAmount;
27        securityDeposit = _securityDeposit;
28        termStartDate = _termStartDate;
29        termEndDate = _termEndDate;
30        terminated = false;
31    }
32
33    function payRent() external payable {
34        require(msg.sender == tenant, "Only tenant can
35        pay rent");
36        require(msg.value == rentAmount, "Incorrect rent
37        amount");
38        emit RentPaid(msg.value);
39    }
40
41    function paySecurityDeposit() external payable {
42        require(msg.sender == tenant, "Only tenant can
43        pay security deposit");
44        require(msg.value == securityDeposit, "Incorrect
45        security deposit amount");
46        emit SecurityDepositPaid(msg.value);
47    }
48
49    function terminateContract() external {
50        require(msg.sender == landlord || msg.sender ==
51        tenant, "Unauthorized");
52        terminated = true;
53        emit ContractTerminated(msg.sender);
54    }
55
56    function getContractDetails() external view returns (
57        address,
58        address,
59        string memory,
60        uint256,
61        uint256,
62        uint256,
63        uint256,
64        bool
65    ) {
66        return (landlord, tenant, propertyAddress,
67            rentAmount, securityDeposit, termStartDate,
68            termEndDate, terminated);
69    }
70 }

```

**Listing 2: Solidity Smart Contract for Rental Agreement**

This proposed validation approach certifies that the smart contract accurately shows the intentions outlined in its e-contract.

By using knowledge graphs, we enhance the reliability of automated smart contract execution systems that builds trust among stakeholders in smart contract integrity.

## 5 Experimental Results

In this section, we discuss with a working example and show the results generated by our proposed approach.

### 5.1 Working Example

An e-contract, or electronic contract, is a digital version of a traditional agreement between two or more parties. In the rental agreement example, as shown in Listing 1, the landlord and tenant agree to specific terms such as the rental period, monthly rent, and responsibilities like paying utilities and maintaining the property. In place of signing a traditional contract on paper approved by court of law, both parties can review and approve the contract electronically, making the process faster and more convenient.

A smart contract takes this step further by using computer code to enforce the agreement automatically. The rental agreement is written using a programming language called Solidity and deployed on a blockchain, as shown in Listing 2. In this version, the contract stores all key details, such as the landlord's and tenant's addresses, the rent amount, the security deposit, and the rental period. Once the smart contract is created, it acts like a digital agent that can receive rent payments, hold deposits, and track whether the contract has been ended.

The smart contract also includes built-in rules that ensure only the tenant can make payments and that the correct amount is sent. It keeps track of payments and emits events—like a receipt—when rent or a deposit is paid. Importantly, it also allows either party to end the contract by calling a function. All actions taken on the contract are recorded on the blockchain, which means they are secure and transparent and cannot be changed later without everyone's knowledge.

This setup improves trust between the landlord and tenant since the rules are programmed and cannot be altered unfairly. There's no need for a middleman like a rental agency to manage the process. Smart contracts also reduce the chance of disputes, as everything is recorded clearly and handled automatically. As more people use blockchain in everyday transactions, smart contracts are becoming a reliable and efficient way to manage digital agreements.

Using the framework for generation of E-Contract Knowledge Graph shown in Fig. 1 on E-Contract shown in Listing 1, we generate the knowledge graph as shown in Fig. 3. Similarly, using the framework for generation of Smart Contract Knowledge Graph shown in Fig. 2 on Smart Contract shown in Listing 2, we generate the knowledge graph as shown in Fig. 4.

### 5.2 Results

For the validation phase, we process JSON files to compare data extracted from knowledge graphs representing the e-contract and the smart contract. Initially, we extracted pertinent information from each knowledge graph, encapsulating entities, relationships, and other relevant details. We then compared the JSON files generated from the two knowledge graphs. This comparative analysis

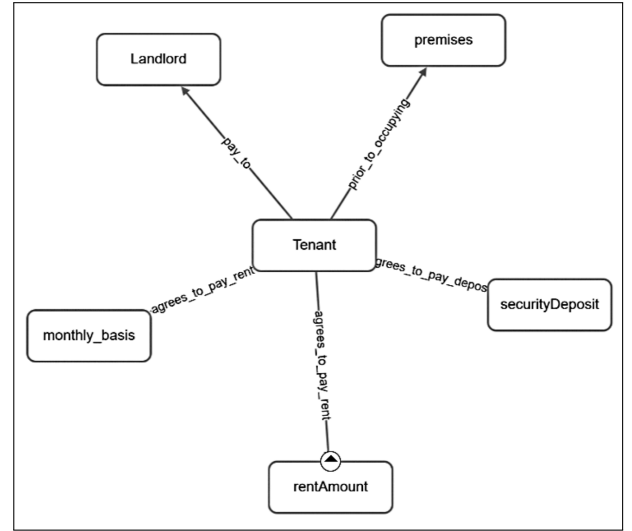


Figure 3: Knowledge graph of e-contract

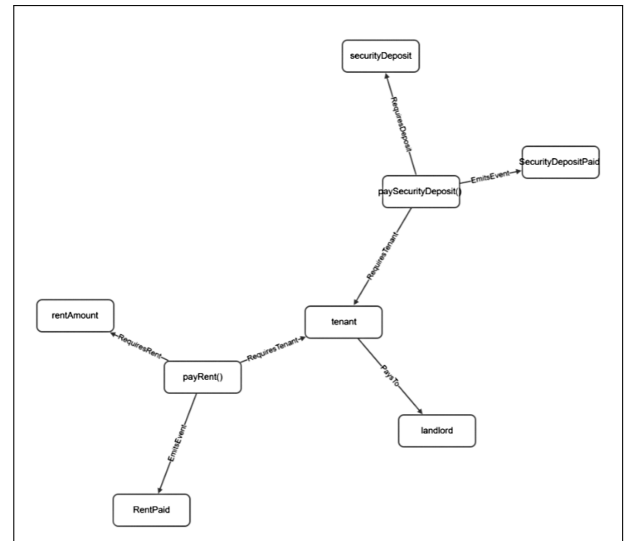


Figure 4: Knowledge graph of Smart Contract

aimed to discern discrepancies, differences, or inconsistencies between the contractual obligations outlined in the e-contract and their representation in the smart contract [7].

Through this comparison process, we meticulously scrutinised whether the JSON files were identical, highlighting any variations between them. Subsequently, we meticulously examined the results to pinpoint any missing terms or entities in the e-contract or the smart contract knowledge graph, as shown in Fig. 6. We compare both the knowledge graphs generated for E-Contract and Smart Contract as shown in Fig. 5, and observe that most of the entities present in E-Contract are available in Smart Contract with the relationships. We have shown the mapping with a dashed line (---) connecting the same entities.

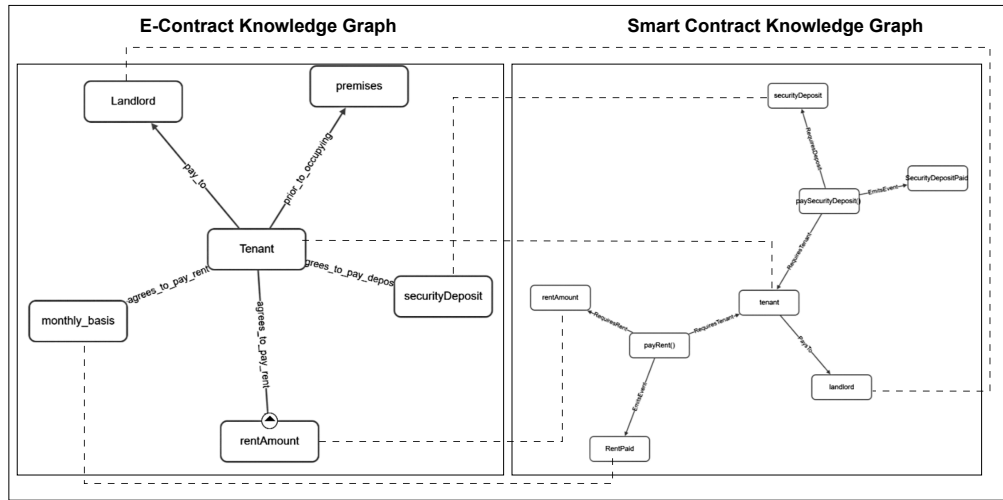


Figure 5: Validation of generated knowledge graphs

```
@VirtualBox:~/final_project$ python3 validation.py
{'rentamount', 'premises', 'tenant', 'monthly_basis', 'securitydeposit', 'landlord'}

{'rentamount', 'tenant', 'rentpaid', 'paysecuritydeposit()', 'securitydepositpaid', 'securitydeposit', 'landlord', 'payrent()'}

missing strings
['premises', 'monthly_basis']
@VirtualBox:~/final_project$
```

Figure 6: Results after the comparison of json files

## 6 Conclusion and Future Work

In this paper, we addressed the challenge of aligning e-contracts written in natural language with smart contracts written in solidity language. We focused on analyzing and comparing these contract types using advanced preprocessing, dependency parsing, and knowledge graph construction techniques. By extracting entities and building knowledge graphs, we aimed to identify discrepancies or differences in representation between e-contracts and smart contracts.

In the future, this work could be expanded by converting e-contracts into Abstract Syntax Trees (ASTs) and comparing them with the ASTs of their related smart contracts. This would involve creating programs that can read and break down e-contracts into structured forms, similar to how smart contracts are represented. After generating both the versions in AST format, they can be validated to see if the smart contract accurately reflects the terms written in the e-contract. Adding this kind of comparison would make the whole process of validating digital contracts more thorough and trustworthy.

## Acknowledgement

This work is sponsored under the grant DIAL Scheme, DST, Government of India given to NITMINER Tech. Pvt. Ltd. All rights reserved between NITMINER Tech. Pvt. Ltd. and IBITF, IIT Bhilai, Government of India.

## References

- [1] Muhamad Abdurahman, Fariz Darari, Hans Lesmana, Muhtar Hartopo, Immanuel Rhesa, and Berty Chrismartin Lumban Tobing. Lex2kg: automatic conversion of legal documents to knowledge graph. In *2021 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 1–5. IEEE, 2021.
- [2] Menaka Pushpa Arthur. Automatic source code documentation using code summarization technique of nlp. *Procedia Computer Science*, 171:2522–2531, 2020.
- [3] Third International Conference on Computing and Network Communications (CoCoNet'19).
- [4] E Chen, Bohan Qin, Yan Zhu, Weijing Song, Shengdian Wang, Cheng-Chung William Chu, and Stephen S Yau. Spesc-translator: Towards automatically smart legal contract conversion for blockchain-based auction services. *IEEE Transactions on Services Computing*, 15(5):3061–3076, 2021.
- [5] Sangharatna Godbole and P. Radha Krishna. Sol-repairer: Solidity smart contract dead code repairer. In *Proceedings of the 18th Innovations in Software Engineering Conference, ISEC '25*, New York, NY, USA, 2025. Association for Computing Machinery.
- [6] Sengharatna Godbole and P. Radha Krishna. Poster: Verisol-mce: Verification-based condition coverage analysis of smart contracts using model checker engines. In *2024 IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 434–437, 2024.
- [7] Aric Hagberg, Pieter J Swart, and Daniel A Schult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Laboratory (LANL), Los Alamos, NM (United States), 2008.
- [8] Daniele Magazzeni, Peter McBurney, and William Nash. Validation and verification of smart contracts: A research agenda. *Computer*, 50(9):50–57, 2017.
- [9] Kaiz Merchant and Yash Pande. Nlp based latent semantic analysis for legal text summarization. In *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1803–1807, 2018.
- [10] Mariana Y. Noguti, Eduardo Vellasques, and Luiz S. Oliveira. Legal document classification: An application to law area prediction of petitions to public prosecution service. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020.
- [11] Chaochen Shia, Yong Xiang, Jiangshan Yub, Longxiang Gaoa, and Keshav Soodc. Solctrans: Towards machine translation of solidity smart contract source code. *CoRR*, 2021.
- [12] Kritagya Upadhyay, Ram Dantu, Zachary Zaccagni, and Syed Badruddoja. Is your legal contract ambiguous? convert to a smart legal contract. In *2020 IEEE International Conference on Blockchain (Blockchain)*, pages 273–280. IEEE, 2020.
- [13] Maximilian Wohrer and Uwe Zdun. Smart contracts: security patterns in the ethereum ecosystem and solidity. In *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 2–8. IEEE, 2018.