

ARTICLE

CISG Smart Contracts: AI Development and Blockchain Validation

Hatem Mabrouk*

Abstract

Ensuring legal compliance in smart contracts is a critical challenge for the widespread adoption of blockchain technology. While smart contracts offer automation and efficiency, they often fail to align with established regulatory frameworks. This misalignment can lead to disputes, financial losses, and reduced trust in decentralized systems. Existing research has focused primarily on the technical and security aspects of smart contracts, with insufficient emphasis on ensuring their legal compliance. The contribution of the present article is to develop and validate a novel plugin, based on artificial intelligence (AI), trained on the United Nations Convention on Contracts for the International Sale of Goods (CISG) provisions to automate the creation of legally compliant blockchain-based smart contracts. This plugin translates CISG requirements into actionable smart contract logic, generating executable Solidity code. These contracts are then tested on Ethereum blockchain platforms to ensure both legal compliance and technical functionality. By integrating CISG rules into the automated generation process, this methodology reduces the manual effort required for contract drafting and validation while minimizing the risk of non-compliance. The results demonstrate that AI-driven tools can enhance contract automation, ensuring enforceability while reducing the risk of non-compliance. However, the findings highlight that AI cannot fully replace legal professionals, as human expertise remains essential for interpreting contractual disputes and ensuring compliance with complex legal frameworks. Nevertheless, this research provides a robust foundation for future advancements in automating legal processes and promoting trust in decentralized ecosystems.

Keywords: AI, legal validation, smart contracts, blockchain, compliance.

* Hatem Mabrouk is a Professor of International Business at Tecnológico de Monterrey, Mexico, who has authored numerous books and publications at the intersection of technology and law. He is the Blockchain Leader at the EGADE Business School Center of Digital Evolution, the founder of FMTO.com, and holds advanced degrees in law and international business. He is an active member of multiple professional organizations and has led blockchain-based projects for several companies. ORCID: <https://orcid.org/0000-0001-5816-7290>.

A Introduction

Blockchain technology has revolutionized various industries by introducing smart contracts—self-executing digital agreements stored on the blockchain. These contracts automate processes, eliminate intermediaries, and enhance transparency. Despite their transformative potential, smart contracts face significant challenges, particularly in ensuring compliance with legal standards and regulatory frameworks. Misalignments in this regard can lead to disputes, undermining trust and limiting the adoption of smart contracts in industries requiring legal validity.¹ Addressing these gaps is critical for realizing blockchain's full potential in a legally sensitive sectors. Legal compliance remains a critical yet underexplored domain in the evolution of smart contracts. Current research extensively explores the technical and security aspects of smart contracts, such as their automation and reliability, but often neglects their enforceability under regulatory frameworks.² This oversight has resulted in ambiguities and vulnerabilities that can hinder smart contracts' utility in international trade.³

This study proposes a novel approach to bridge the gap between technical functionality and legal compliance by leveraging AI. Specifically, it involves creating a plugin trained on the CISG framework to generate a legally compliant Solidity smart contract. The proposed methodology tests this smart contract on the Ethereum blockchain to validate both its technical and regulatory alignment. The contribution of this research lies in automating the translation of legal provisions into executable smart contracts, thus ensuring compliance while reducing manual drafting efforts.

By focusing on the CISG framework, this study demonstrates how international trade standards can be integrated into smart contracts. This approach not only aligns the technical logic of smart contracts with legal intent but also provides a scalable solution for deploying blockchain technologies in global trade, finance, and other industries requiring high levels of legal certainty.⁴

B Literature Review

The application of AI in smart contract validation has been a key area of exploration. Barboni et al. (2022, p. 23)⁵ discuss the challenges associated with testing

- 1 Louati, H., et al. (2024). *Adopting Artificial Intelligence to Strengthen Legal Safeguards in Blockchain Smart Contracts*. *Journal of Theoretical and Applied Electronic Commerce Research*, p. 2142. <https://doi.org/10.3390/jtaer19030104>
- Duchmann, F., & Koschmider, A. (2019). *Validation of Smart Contracts Using Process Mining (short paper)*, p. 14.
- 2 Patel, O. (2024). *AI-Driven Smart Contracts*. *Journal of Artificial Intelligence & Cloud Computing*, p. 3. [https://doi.org/10.47363/jaicc/2024\(3\)e120](https://doi.org/10.47363/jaicc/2024(3)e120)
- 3 Clack, C. (2018). *Smart Contract Templates: Legal semantics and code validation*. *Journal of Digital Banking*, p. 340. <https://doi.org/10.69554/xyvt7035>
- 4 Tolmach, P., et al. (2020). *A Survey of Smart Contract Formal Specification and Verification*. *ACM Computing Surveys (CSUR)*, p. 4. <https://doi.org/10.1145/3464421>
- Barboni, M., et al. (2022). *Smart Contract Testing: Challenges and Opportunities*. *WETSEB*, p. 23. <https://doi.org/10.1145/3528226.3528370>
- 5 Barboni, M., et al. (2022). *Smart Contract Testing: Challenges and Opportunities*. *WETSEB*, p. 23. <https://doi.org/10.1145/3528226.3528370>

smart contracts, including the complexity of blockchain environments and insufficient pre-release validation mechanisms. They highlight the importance of automated tools in identifying vulnerabilities and ensuring the robustness of contract execution. Similarly, Tong et al. (2022, p. 22)⁶ propose using AI-driven segmentation techniques to transform natural language agreements into executable smart contracts, thus reducing manual errors.

Several studies also emphasize formal methods for ensuring smart contract correctness. Singh et al. (2020, p.5)⁷ and Godoy et al. (2022, p. 292)⁸ advocate for using theorem proving, model checking, and predicate abstractions to detect and mitigate vulnerabilities in smart contracts. These approaches enhance the functional accuracy and security of blockchain-based agreements.

I Legal Compliance and Smart Contracts

Ensuring compliance with legal standards is one of the most critical yet underexplored areas of smart contract development. Clack (2018, p. 340)⁹ introduces the concept of “Smart Contract Templates,” which utilize standardized code to translate legal agreements into enforceable digital contracts. Wang et al. (2019, p. 2270)¹⁰ further provide a six-layer architecture for aligning blockchain-enabled contracts with regulatory requirements, offering a systematic framework to bridge the gap between technology and legal enforceability.

In the context of international trade, Louati et al. (2024, p. 2142)¹¹ propose AI-powered frameworks to strengthen legal safeguards in blockchain smart contracts, highlighting the importance of aligning smart contract logic with international legal standards. Additionally, Zou et al. (2021, p. 45)¹² identify key challenges developers face, including insufficient tools, unclear methodologies, and the difficulty of integrating legal frameworks into blockchain systems.

II Advancements in Smart Contract Technology

Recent advancements in AI and blockchain have introduced innovative mechanisms for improving the functionality and compliance of smart contracts. Comuzzi et al. (2022, p. 57)¹³ propose quality-aware transaction validation techniques to

- 6 Tong, Y., et al. (2022). *Smart Contract Generation Assisted by AI-Based Word Segmentation*. *Applied Sciences*, p. 22. <https://doi.org/10.3390/app12094773>.
- 7 Singh, A., et al. (2020). *Blockchain Smart Contracts Formalization: Approaches and Challenges to Address Vulnerabilities*. *Computers & Security*, p. 5. <https://doi.org/10.1016/j.cose.2019.101654>.
- 8 Godoy, J., et al. (2022). *Predicate abstractions for smart contract validation*. *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, p. 292. <https://doi.org/10.1145/3550355.3552462>.
- 9 Clack, C. (2018). *Smart Contract Templates: Legal semantics and code validation*. *Journal of Digital Banking*, p. 340. <https://doi.org/10.69554/xyvt7035>.
- 10 Wang, S., et al. (2019). *Blockchain-Enabled Smart Contracts: Architecture, Applications, and Future Trends*. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, p. 2270. <https://doi.org/10.1109/TSMC.2019.2895123>.
- 11 Louati et al., 2024, see footnote 1.
- 12 Zou, W., et al. (2021). *Smart contract development: Challenges and opportunities*. *IEEE Transactions on Software Engineering*, 47(10), 2084–2106, p. 45. <https://doi.org/10.1109/TSE.2019.2942301>
- 13 Comuzzi, M., et al. (2022). *Toward Quality-Aware Transaction Validation in Blockchains*. *IEEE Software*, p. 57. <https://doi.org/10.1109/ms.2022.3152176>.

prevent poor data quality from affecting smart contract performance. Similarly, Krichen (2023, p. 18)¹⁴ discusses the use of AI algorithms to enhance the security of blockchain transactions, addressing key vulnerabilities in contract execution.

Runtime validation has also emerged as a critical component of smart contract reliability. Li et al. (2020, p. 442)¹⁵ introduce "Solythesis," a Solidity compiler with integrated runtime validation to detect and reject transactions violating predefined invariants. This approach ensures compliance during contract execution, significantly reducing the risks of errors or fraud.

III Challenges and Future Directions

Despite these advancements, significant challenges persist in achieving legally compliant and technically robust smart contracts. Duchmann and Koschmider (2019, p. 14)¹⁶ emphasize the role of process mining in identifying compliance gaps in smart contracts, while Tolmach et al. (2020, p. 4)¹⁷ argue for integrating formal specifications into smart contract design to address these issues systematically.

Moreover, existing development tools remain rudimentary, and resource limitations often constrain blockchain environments.¹⁸ Future research must focus on integrating AI tools with regulatory frameworks to ensure that smart contracts meet both technical and legal requirements. By bridging this gap, researchers can enable broader adoption of smart contracts across global industries.

C Methodology

This study employs a structured methodology to develop, train, and evaluate an artificial intelligence (AI)-powered plugin capable of generating legally compliant smart contracts from uploaded CISG-based legal contracts. The research is guided by the following two questions: "1) *Can an AI-powered plugin trained on the CISG framework accurately generate a CISG-compliant contract and convert it into legally compliant smart contracts?* 2) *Can these contracts be successfully deployed and executed on a blockchain?*" Each step in the methodology is designed to address these questions.

The process begins with the development and training of the plugin, where the full text of the CISG and annotated examples of compliant contracts are uploaded. Using advanced natural language processing (NLP) techniques, the plugin learns to extract key legal clauses, terms, and obligations from the CISG framework. Iterative testing ensures the plugin's accuracy in interpreting and applying these principles to generate structured outputs.

Once trained, the plugin is asked to generate a legal contract based on the CISG. The contract is then uploaded back to the plugin to be analyzed and to verify

14 Krichen, M. (2023). *Strengthening the Security of Smart Contracts through the Power of Artificial Intelligence*. *Comput.*, 12, 107, p. 18. <https://doi.org/10.3390/computers12050107>.

15 Li, A., et al. (2020). *Securing smart contract with runtime validation*. *PLDI '20*, p. 442. <https://doi.org/10.1145/3385412.3385982>.

16 Duchmann & Koschmider, 2019, see footnote 1.

17 Tolmach et al., 2020, see footnote 4.

18 Zou et al., 2021, see footnote 12; Barboni et al., 2022, see footnote 5.

the contract’s compliance with the CISG and the annotated examples of CISG-compliant contracts. Then, the plugin generates a corresponding smart contract in Solidity, translating the legal provisions into executable blockchain code. This step directly evaluates the plugin’s ability to bridge the gap between traditional legal documentation and blockchain-based automation.

The generated smart contract is tested in Remix, an Ethereum-based Integrated Development Environment (IDE). This phase includes validating the syntax and logic of the smart contracts, simulating a real-world scenario to assess runtime behavior, and ensuring that the operational logic aligns with the original legal contract. These tests provide critical insights into whether the plugin-generated contract meets both legal and functional requirements.

Based on the results of the testing phase, the final version of the smart contract is deployed to the Ethereum blockchain, where its performance is monitored in live environments. This deployment phase ensures that the smart contract can operate as intended and meet both technical and legal standards, thereby addressing the core research questions.

This methodology systematically addresses the research questions by evaluating whether the plugin can effectively generate and convert a CISG-based legal contract into a functional and deployable smart contract. By focusing on this capability, the study provides a scalable framework for automating legal-to-smart contract translation in global trade and other legally sensitive domains. Below is the table summarizing the overall methodology outlined in this research, while the subsequent Data Collection section provides a more detailed breakdown of its implementation phases.

Based on Table 1, this study is applied, qualitative, exploratory, and in a developmental manner. It focuses on creating a practical solution – an AI-powered

Table 1 Methodology for Developing and Testing the CISG Plugin for Smart Contract Generation

Step	Description	Objective
Plugin Development	Develop an AI-powered plugin by training it on the full text of the CISG using NLP techniques. Annotated examples of CISG-compliant contracts are included for training.	To enable the plugin to understand CISG principles and legal clauses for generating smart contracts.
Plugin Training and Validation	Train and test the plugin to ensure it can accurately extract and replicate key legal clauses, terms, and logic from the CISG. Iterative testing ensures its functionality.	To evaluate the plugin’s ability to interpret a specific CISG-based legal contract and extract actionable data.
Legal Contract Generation, Upload and Analysis	The trained plugin generates a legal contract based on the CISG framework. The contract is then uploaded back to the plugin to be analyzed and to verify the contract’s compliance with the CISG and the annotated examples of CISG-compliant contracts.	To evaluate the plugin’s ability to generate a CISG-based legal contract.

Smart Contract Generation	The plugin generates a Solidity smart contract from the uploaded CISG-based legal contract. This includes translating clauses like payment terms, delivery obligations, and dispute resolution into executable code.	To assess the plugin’s capability to convert a legal contract into a functional smart contract.
Testing in Remix	Test the generated smart contract in Remix by compiling it, to validate the syntax, logic. Smart contract compilation is the process of converting human-readable code into machine-readable instructions so that it can run on a blockchain.	To verify the functional accuracy and executable nature of the smart contract in a blockchain environment.
Deployment to Ethereum	Deploy the finalized smart contracts to the Ethereum blockchain. Monitor their behavior in live environments and resolve any unexpected issues.	To confirm that the generated smart contracts are deployable and perform as expected in real-world scenarios.

plugin – to address the gap between legal contracts and smart contracts. The research bridges technological development and legal integration by employing experimentation and iterative validation without relying on statistical analysis or mathematical modelling. Instead, the focus is on testing real-world applicability and functionality in blockchain environments.

D Data Collection

The data collection process was structured into six phases to systematically develop and validate a ChatGPT-powered plugin trained on the CISG. These phases ensured that the plugin could accurately interpret, generate, and validate CISG-compliant contracts.

I Phase 1: Plugin Development

The first phase of data collection involved training a ChatGPT-powered plugin on the full text of the CISG and relevant legal frameworks. Key resources included the CISG text as the primary legal framework and the International Trade Centre’s *Model Contracts for Small Firms: Legal Guidance for Doing International Business*¹⁹ to provide practical examples. Carefully annotated legal texts enabled the AI model to recognize key contractual elements such as payment obligations, delivery conditions, risk transfer, and dispute resolution mechanisms. The goal of this phase was to equip the plugin with the foundational knowledge of the CISG and its application in international trade contracts. The plugin can be accessed from ChatGPT’s “Explore GPTs” menu.²⁰

19 International Trade Centre. (2010). *Model contracts for small firms: Legal guidance for doing international business*. International Trade Centre. <https://www.intracen.org/file/modelcontractspdf-0>.
20 Mabrouk, H. (2025a). *CISG Smart Contract Advisor* [ChatGPT plugin]. <https://chat.openai.com/g/g-67930eed6d408191abd0abe856b7bbe5>.

II Phase 2: Plugin Training and Validation

In this phase, the plugin was tested to evaluate its ability to extract, replicate, and generate CISG-compliant contracts. The AI model was refined through iterative testing, where it extracted key legal clauses, terms, and logic from the CISG. Automated contract structuring ensured that the generated contracts aligned with CISG provisions. Enhancing AI understanding by eliminating extraneous elements such as "short version" and "long version" from contracts helped streamline the model's outputs. Through this process, the AI model improved its ability to recognize recurring patterns in contractual language and automate the structuring of legal agreements with consistency and accuracy. A sample of a generated contract can be downloaded from GitHub.²¹

III Phase 3: Legal Contract Upload and Analysis

In this phase, the generated CISG-based contract was uploaded back to the plugin to be analyzed and to evaluate its ability to interpret specific legal contracts based on the CISG framework. The plugin assessed the contracts to extract actionable legal data, identifying key contractual obligations and risk allocations. Additionally, it verified CISG compliance, ensuring the generated contracts were legally valid and technically accurate.

Figure 1 illustrates the verification results of an AI-generated CISG-compliant contract. This assessment confirmed that the contract adhered to CISG provisions, properly supplemented with Swiss law where necessary, and included dispute resolution mechanisms. This structured data collection approach ensured that the plugin could generate, refine, and validate CISG-based contracts with increasing efficiency, reinforcing its compliance with international commercial sale regulations.

IV Phase 4: Smart Contract Generation:

The process of ensuring that the Solidity-based smart contract accurately reflected the CISG-based legal contract and remained CISG-compliant required multiple iterative verification steps. Initially, the smart contract captured fundamental contractual elements such as parties, goods, payment terms, and delivery obligations. However, the first verification revealed several gaps, including the absence of an inspection period, incomplete dispute resolution mechanisms, missing force majeure provisions, and a lack of proper handling for late payment and late delivery penalties. Additionally, while the legal contract specified payment via a Letter of Credit (L/C), the smart contract only allowed on-chain transactions, requiring an update to support off-chain L/C verification.

Through successive iterations, the contract was refined to align fully with CISG provisions. A 5-day inspection window was introduced to allow the buyer to verify the delivered goods before finalizing the contract. During this time, the payment was still held in the smart contract or L/C. The dispute resolution mechanism was

21 Mabrouk, H. (2025b). *AI-generated CISG contract*. GitHub. <https://github.com/hatmabrouk/CISG-Smart-Contracts/blob/main/AI-Generated%20CISG%20Contract.docx>.

Figure 1 CISG Compliance Verification by AI

✓ Arbitration Clause:

- **UNCITRAL arbitration in Geneva, Switzerland, ensuring neutral dispute resolution.**

Final Verdict: ✓ Fully CISG-Compliant

The **Modified CISG Contract** is **fully compliant** with **CISG** provisions.

- ✓ No conflicts with CISG rules.
- ✓ Properly supplements CISG with **Swiss law** where necessary.
- ✓ Provides clear dispute resolution and remedies.

This contract is enforceable under CISG with no necessary further modifications.

enhanced to require a structured reason for any disputes, ensuring clarity and adherence to international arbitration principles. For example, calling the dispute resolution function can no longer take place once the contract is completed. Late payment and late delivery penalties were implemented to mirror the 0.5% per day penalty structure outlined in the legal contract. Furthermore, a *force majeure* clause was integrated to allow for contract suspension in the event of unforeseen circumstances. Finally, an off-chain payment verification function was added, allowing future integration with an L/C verification system. After these refinements, a final verification confirmed that the Solidity contract now fully encapsulated all terms and conditions of the CISG-based legal contract, ensuring both legal enforceability and compliance with international trade standards. The finalized smart contract can be retrieved from GitHub.²²

V Phase 5: Testing in Remix

This phase involves the step of compiling the smart contract on Remix IDE. Smart contract compilation is the process of converting human-readable code into machine-readable instructions so that it can run on a blockchain.

²² Mabrouk, H. (2025c). *CISGSaleContract - CISG.sol*. GitHub. <https://github.com/hatmabrouk/CISG-Smart-Contracts/blob/main/CISGSaleContract%20-%20CISG.sol>.

Figure 2 Contract Compilation

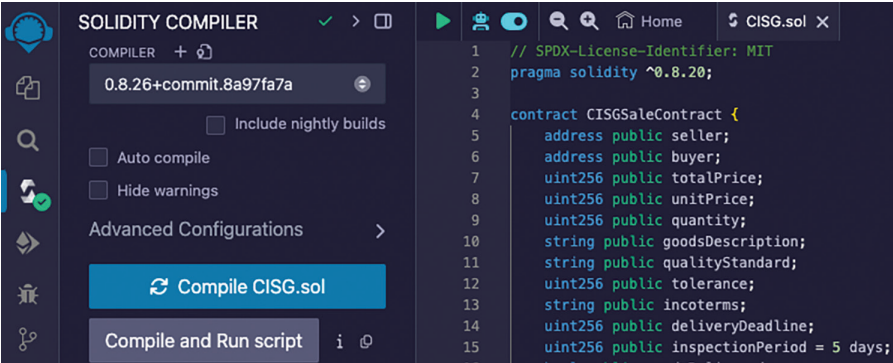


Figure 2 reflects the successful compilation of the Solidity-based smart contract, which marks a significant milestone in ensuring CISG compliance within an automated framework. This achievement confirms the smart contract's structure, including its core. The green check mark following the smart contract compilation process indicates a successful transformation of the high-level Solidity code into low-level bytecode, which can be executed on the Ethereum Virtual Machine (EVM). This signifies that the contract is syntactically correct, free of critical compilation errors, and ready for deployment on the blockchain. A successful compilation ensures that the contract's logic has been properly encoded, facilitating its subsequent execution and interaction within the decentralized environment.

VI Phase 6: Deployment

Deploying a smart contract in Remix IDE requires providing initial parameters that configure the contract according to the business and legal requirements.

Figure 3 shows the deployment interface of the smart contract “CISGSaleContract”, where various fields must be populated before deployment. Below is a breakdown of each section and an explanation of its significance within the contract execution.

1 Contract Selection & EVM Version

At the top of Figure 3, the *Contract* field displays “CISGSaleContract - CISG.sol”, indicating that the correct smart contract file is selected. The EVM version is set to Cancun, ensuring compatibility with the latest Ethereum EVM updates at the time of writing. Ensuring the correct EVM version prevents unexpected errors due to opcode changes.

Figure 3 Contract Deployment

CONTRACT

CISGSaleContract - CISG.sol

evm version: cancun

DEPLOY

_BUYER:

0x29cac812614a1aaa644932

_TOTALPRICE:

1500000000000000000

_UNITPRICE:

1500000000000000000

_QUANTITY:

1000

_GOODSDESCRIPTION:

"Industrial-grade steel"

_QUALITYSTANDARD:

"ISO 9001 Certified"

_TOLERANCE:

2

_DELIVERYDEADLINE:

2592000

Calldata

Parameters

transact

2 Buyer Address (*_BUYER*)

This field requires an Ethereum address for the buyer “(*_BUYER*)”. This input represents the counterparty to the contract who is responsible for making the payment through an L/C. The entered address:

(0x29cac812614a1aaa644932C79E5f316D2c021CE0)

is an example of a valid Ethereum address, which ensures that the contract correctly identifies the buyer within the blockchain network. In the smart contract, the buyer’s address is stored as:

```
address public buyer;
```

This ensures that only the designated buyer can perform actions such as verifying the L/C and accepting the goods.

3 *Total Price* (*_TOTALPRICE*)

The “*_TOTALPRICE*” field requires the total contract price, which is set to 15000000000000000000, i.e. 1.5 ETH in wei. Wei is the smallest unit of Ether (ETH), the cryptocurrency used on the Ethereum blockchain. Just like cents in a dollar or pennies in a pound, Wei allows for very precise transactions by breaking down ETH into tiny fractions. One Ether equals one quintillion Wei (1 ETH = 1,000,000,000,000,000,000 Wei). This small unit helps ensure accuracy in transactions, especially when dealing with smart contracts and micro-payments. The 1.5 ETH value represents the agreed-upon total price of the goods, but it does not involve an actual ETH transfer, since payment is handled off-chain via an L/C. In the smart contract, this value is used for penalty calculations for late delivery, interest calculations for late payments, and contract enforcement to ensure the correct financial agreement is adhered to. The total price is stored in the contract as:

```
uint256 public totalPrice;
```

This ensures that penalties and damages can be calculated if contract obligations are not met.

4 *Unit Price* (*_UNITPRICE*)

The “*_UNITPRICE*” field requires the unit price of the goods per quantity, which is set to 15000000000000000 (0.0015 ETH in wei). This field ensures that price calculations remain accurate. Unit price is defined in the smart contract as:

```
uint256 public unitPrice;
```

This allows for precise tracking of the cost per unit of goods.

5 *Quantity of Goods* (*_QUANTITY*)

The “*_QUANTITY*” field specifies the number of goods involved in the transaction. In Figure 3, this value is 1000, meaning the contract will track and enforce obligations for 1,000 units of goods. This quantity of goods value is stored in:

```
uint256 public quantity;
```

The value ensures price calculations, penalties, and obligations are correctly aligned with the contractual agreement.

6 *Goods Description* (*_GOODSDESCRIPTION*):

This field describes the nature of the goods being sold. The entered value, “Industrial-grade steel sheets”, ensures that the contract identifies the type of goods involved. Goods description is stored in:

```
string public goodsDescription;
```

This prevents disputes over the nature and specifications of the goods.

7 *Quality Standard* (*_QUALITYSTANDARD*)

This field ensures that the goods meet a specific standard. The entered value, "ISO 9001 Certified", confirms that the goods must comply with the ISO 9001 quality management system. The quality standard is defined in the contract as:

string public qualityStandard;

This helps enforce contractual quality requirements and buyer rights to reject non-conforming goods.

8 *Tolerance Level* (*_TOLERANCE*)

The "*TOLERANCE*" field specifies the acceptable deviation in the quantity or quality of the goods. The entered value 2 represents a $\pm 2\%$ tolerance, meaning slight deviations from the agreed specifications are permissible. The tolerance level is defined as:

uint256 public tolerance;

This ensures that minor variances do not trigger contract breaches, preventing unnecessary disputes.

9 *Delivery Deadline* (*_DELIVERYDEADLINE*)

This field specifies the period within which the seller must ship the goods. The entered value 2592000 represents 30 days (in seconds) from the contract deployment. This deadline is enforced in:

uint256 public deliveryDeadline;

If the seller fails to ship the goods on time, a late penalty is applied. If delivery does not happen at all, the buyer may declare contract avoidance.

10 *Value Field* (*VALUE*)

The "*VALUE*" section in Figure 3 is set to 0 Wei, which means no ETH is being transferred during deployment. This is correct because the contract does not require an upfront ETH payment. Instead, payment verification happens later through L/C verification. If this field contained ETH, the transaction would be rejected because the constructor does not accept direct ETH payments.

11 *Deploying the Contract* (*TRANSACT Button*)

Once all values are correctly entered, the "*Transact*" button is clicked to deploy the contract on the blockchain. This process writes the contract data to the blockchain, creates a unique contract address for further interactions, and locks the agreed parameters (price, buyer, goods details, and deadlines). After deployment, the buyer must verify the L/C by calling:

```
verifyLetterOfCredit(true);
```

This ensures that the transaction remains aligned with CISG compliance before shipment.

Figure 3 captures the critical pre-deployment setup for the *CISGSaleContract*, ensuring that all required contractual parameters are entered correctly. Unlike traditional smart contracts that handle payments directly in ETH, this contract adheres to CISG principles by requiring an off-chain L/C verification before the transaction can proceed. With this setup, no goods can be shipped unless the buyer confirms the L/C, ensuring compliance with international trade laws and preventing financial risks for the seller. The next step after deployment is for the buyer to verify the L/C, allowing the seller to mark goods as shipped and complete the transaction legally.

The CISG Sale Contract was successfully deployed on the Sepolia testnet, a widely used Ethereum test network that allows developers to simulate smart contract execution before deploying on the mainnet. The deployment process involved inputting the required contract parameters, including the buyer's address, total price, unit price, quantity of goods, and delivery deadlines. Once the deployment transaction was submitted, the process was confirmed on the Ethereum blockchain, and the transaction details can be viewed on Sepolia Etherscan.²³ The deployed smart contract address on the Sepolia network allows for public verification of the contract code and interactions.²⁴ Following this deployment, various contract functions can now be executed, including verifying the L/C, tracking the shipment and acceptance of goods, enforcing penalties for late delivery, and managing disputes.

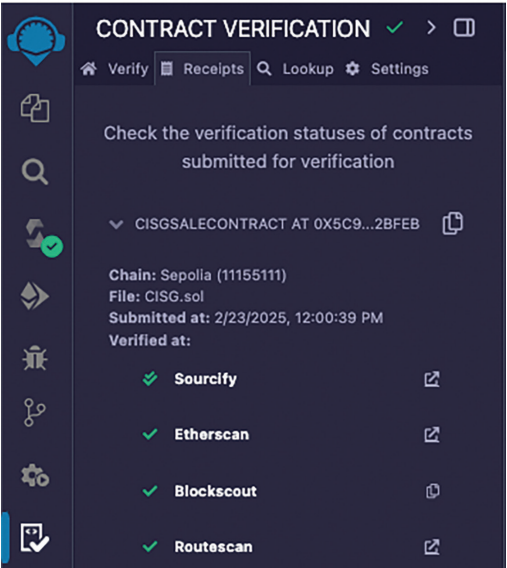
After successfully deploying the "*CISGSaleContract*" on the Sepolia testnet, the contract was verified using the Contract Verification Plugin in Remix. This process ensures transparency and allows the smart contract's source code to be publicly accessible on multiple verification platforms, including Etherscan, Sourcify, Blockscout, and Routerscan, as shown in Figure 4.

Contract verification plays a crucial role in enhancing trust and security in blockchain transactions. By verifying the contract, users can inspect its logic and confirm that it operates as intended without hidden vulnerabilities or malicious code. This transparency is especially important in smart contract applications where financial transactions and legal agreements are automated without intermediaries. Once verified, Etherscan provides an interactive interface that allows users to read and interact with the contract directly, reducing reliance on external tools such as Remix for contract execution.

23 Etherscan. (2025a, February 23). *Ethereum transaction details (Sepolia testnet)*. <https://sepolia.etherscan.io/tx/0x1b2d4a35feb94de50b6330f8c786f74053538e62c8f256ec71a953d4aa5410ae>

24 Etherscan. (2025b, February 23). *Ethereum contract address details (Sepolia testnet)*. <https://sepolia.etherscan.io/address/0x5c94f859cdadd756101ff80f36574f9dacf2bfeb>

Figure 4 *Contract Verification*

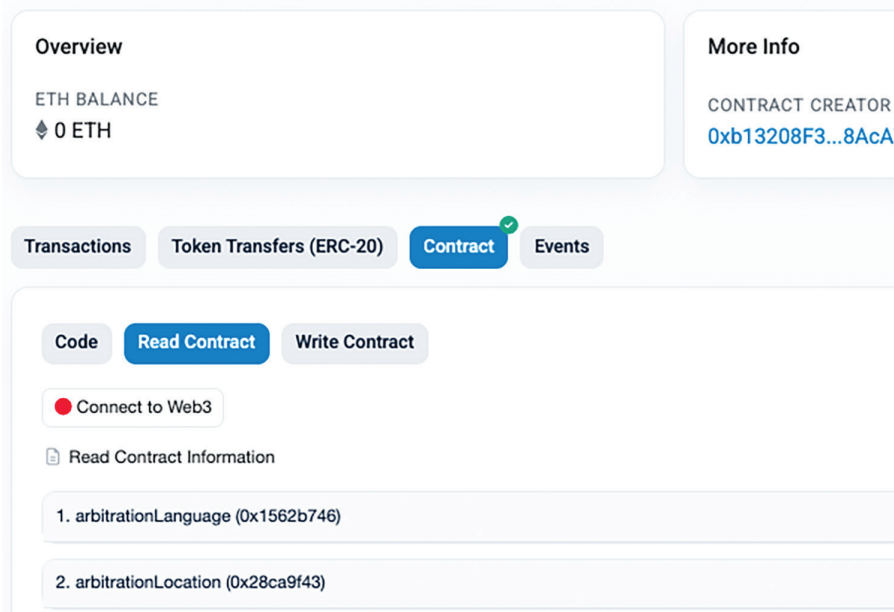


Furthermore, contract verification strengthens security and debugging capabilities. By making the source code publicly available, developers and auditors can analyze the logic, detect potential errors, and ensure compliance with expected behavior. This step is essential in preventing exploits and ensuring that the contract aligns with its intended use case.

The verification process in Remix involved submitting the contract's source code through the Contract Verification Plugin. The tool then compared the deployed bytecode with the provided source code to confirm consistency. Once this process was completed, verification checkmarks appeared on Sourcify, Etherscan, Blockscout, and Routerscan, confirming that the contract is now fully traceable, auditable, and accessible for further interaction on Etherscan. The successful verification, as displayed in Figure 4, demonstrates that the contract's deployment and source code match, ensuring its integrity on the blockchain.

Figure 5 displays the contract "CISGSaleContract" on Etherscan, confirming its successful deployment on the Sepolia testnet. Furthermore, the green check mark pointed at with an arrow in Figure 5 confirms the contract's successful verification. The contract tab allows users to interact with the smart contract directly through the "Read Contract" and "Write Contract" functions. The "Read Contract" section provides access to stored data, such as arbitration details and contract state, while the "Write Contract" section enables authorized parties to execute functions like marking goods as shipped or accepting delivery. This verification ensures that users can independently verify and interact with the contract without relying on external development tools, enhancing transparency and trust in its execution. The next section provides a detailed breakdown of these functions and their role in ensuring a secure and automated international sale agreement.

Figure 5 Smart Contract on EtherScan



E Understanding the Smart Contract Functions

This smart contract governs an international sale of goods under the CISG framework, ensuring compliance with key trade principles such as payment security, delivery tracking, dispute resolution, and contract termination. The contract involves two main parties: the seller, who deploys the contract, and the buyer, whose address is provided at deployment.

1 Functions That Modify the Contract (Action Functions)

Certain functions modify the contract state, meaning they trigger actions such as verifying payment, shipping goods, or resolving disputes. These require explicit execution from either the buyer or the seller.

1 Verifying Payment via L/C

To ensure the contract “CISGSaleContract” functions as intended, users must follow a specific sequence of steps. Initially, the contract is in the *Created* state, requiring the buyer to verify the L/C before any further actions can take place. This verification, executed through the “*verifyLetterOfCredit(bool approved)*” function, is crucial as it validates the payment arrangement and transitions the contract to the “*InProgress state*”. Without this step, the seller is unable to proceed with shipping the goods. Thus, confirming the L/C as approved is a mandatory prerequisite for advancing the transaction.

2 *Marking Goods as Shipped*

Once the buyer has verified the payment by L/C, the seller can confirm that the goods have been shipped by calling *“markGoodsShipped()”*. This function requires that the payment has been approved and that the shipping deadline has not passed. Once executed, the contract updates the status to indicate that the goods are en route, ensuring transparency in the delivery process.

3 *Accepting Goods Upon Delivery*

After the goods are shipped, the buyer must confirm receipt and acceptance by executing *“acceptGoods()”*. This function ensures that the goods were delivered and that the buyer is within the five-day inspection period. If the buyer is satisfied, the function updates the contract state to *“Completed”*, finalizing the transaction.

4 *Applying Late Delivery Penalty*

If the seller fails to deliver goods on time, the buyer can apply a penalty using *“applyLateDeliveryPenalty()”*. This function calculates a daily penalty of 0.5% of the total contract value, as well as interest on overdue payments at an annual rate of 5%. The contract does not automatically penalize the seller; rather, the buyer must execute this function to claim compensation for the delay.

5 *Raising a Dispute*

If disagreements or non-compliance with contract terms arise, either party may invoke the *“raiseDispute(string _reason)”* function. This function documents the dispute, including the reason and timestamp, and updates the contract state to *“Disputed”*, halting further progress until arbitration or resolution occurs. Disputes may arise from issues such as damaged goods, incorrect delivery, or breaches of contract terms. Notably, disputes must be raised before the contract reaches the *“Completed state”*, as once goods are accepted, the contract is finalized, and arbitration is no longer applicable. This underscores the importance of buyers thoroughly inspecting goods before acceptance to preserve their right to dispute.

6 *Invoking Force Majeure (Unforeseen Events)*

If unexpected circumstances such as war, natural disasters, or supply chain disruptions prevent contract execution, either party can pause the contract using *“invokeForceMajeure(uint256 duration)”*. This function temporarily suspends obligations for the specified duration, preventing unfair penalties during periods of force majeure.

7 *Terminating the Contract*

If the contract cannot proceed as planned, either the buyer or seller can execute *“terminateContract(string reason)”*. This function formally cancels the agreement and logs the reason for termination. Once executed, the contract state updates to *“Terminated,”* ensuring clarity in case of contract failure.

8 *Sending Notifications Between Parties*

To facilitate communication, the function “*sendNotification(address recipient, string message)*” allows buyers and sellers to send messages to each other within the contract. These messages are stored on the blockchain and serve as formal notifications regarding contract status updates.

II *Functions That Retrieve Contract Information (Getter Functions)*

The contract also includes getter functions that allow users to view contract details without modifying its state. These functions provide key insights into contract terms, transaction status, and stored data. In simple terms, these functions do not modify the contract but allow users to view contract details.

The getter functions, as displayed in Table 2, provide real-time contract information. These getter functions allow parties to quickly access contract information without the need for additional blockchain transactions, making the system more efficient and transparent.

Table 2 Key Getter Functions

Function Name	Description
<i>arbitrationMethod()</i>	Returns dispute resolution method (<i>ad hoc</i> arbitration under UNCITRAL Rules)
<i>arbitrationLocation()</i>	Returns arbitration location (Geneva)
<i>arbitrationLanguage()</i>	Returns arbitration language (English)
<i>arbitrators()</i>	Returns number of arbitrators (3)
<i>buyer()</i>	Returns buyer's address
<i>seller()</i>	Returns seller's address
<i>state()</i>	Returns current contract state
<i>goodsDescription()</i>	Returns description of goods
<i>qualityStandard()</i>	Returns required quality standard
<i>incoterms()</i>	Returns shipping terms (FOB - London Port)
<i>deliveryDeadline()</i>	Returns delivery deadline in UNIX time
<i>inspectionPeriod()</i>	Returns buyer's inspection window (5 days)
<i>forceMajeureActive()</i>	Checks if force majeure is active
<i>penaltyRate()</i>	Returns penalty rate for late delivery (0.5% per day)
<i>interestRate()</i>	Returns interest rate on late payments (5% annually)
<i>totalPrice()</i>	Returns total contract price
<i>unitPrice()</i>	Returns price per unit of goods

According to the previous explanations, some functions are restricted to specific roles. The buyer is solely responsible for verifying the L/C, applying penalties, and accepting the goods, while the seller is the only party allowed to mark the goods as shipped. Both parties, however, can raise disputes, invoke force majeure, terminate the contract, or send notifications to each other. If the contract is terminated or disputed, no further transactions can occur until resolved. By following this structured process, users can avoid failed transactions and understand the contract's role in ensuring a secure, legally structured international sale of goods.

This smart contract automates international sales agreements by providing a secure, transparent, and legally enforceable way to conduct trade. By integrating payment verification, shipment tracking, penalties, dispute resolution, and force majeure protection, the contract ensures a structured and fair process for both buyers and sellers. Furthermore, the inclusion of getter functions enables real-time access to contract data, making it easy to track the status of a transaction. The contract follows the CISG framework, ensuring that international trade principles are enforced automatically through blockchain technology. This digital approach eliminates the need for intermediaries, reduces fraud risks, and provides tamper-proof transaction records, making it a powerful tool for modern international trade.

F Findings

This research sought to determine whether AI could fully replace lawyers in drafting and validating contracts, particularly in the context of the CISG. The development and deployment of a CISG-compliant smart contract provided a practical framework for evaluating AI's ability to automate legal compliance while ensuring technical functionality on the blockchain. Through iterative development and testing, key insights emerged regarding the strengths and limitations of AI in contract automation.

One of the primary findings is that AI can significantly enhance the efficiency of contract drafting and validation by converting legal clauses into executable smart contracts. The developed AI-powered plugin successfully translated CISG provisions into Solidity code, ensuring that key legal requirements—such as payment verification, delivery tracking, dispute resolution, and penalty enforcement—were incorporated into the contract's logic. This demonstrated AI's potential to automate routine contractual processes while reducing manual effort. Additionally, the integration of legal principles into smart contracts provided a structured and enforceable framework for international transactions, reducing the risk of ambiguity and misinterpretation.

Despite these advancements, the study also revealed the limitations of AI in fully replacing legal professionals. While the AI-powered plugin could accurately generate legally compliant smart contracts based on predefined rules, it struggled with complex, context-dependent legal reasoning. For example, dispute resolution required human interpretation of contractual breaches, and certain contract modifications needed subjective judgment beyond what AI could codify. These

findings suggest that while AI can automate the structural aspects of contract drafting, human expertise remains essential for handling nuanced legal scenarios, particularly in dispute resolution and regulatory interpretation.

Another key observation pertains to the execution of the smart contract and its enforcement on the Ethereum blockchain. The contract was deployed and verified successfully on the Sepolia testnet, demonstrating its operational integrity. Verification on platforms such as Etherscan, Sourcify, Blockscout, and Routerscan ensured transparency and security, allowing users to interact with the contract with confidence. The structured sequence of contract execution—from verifying the L/C to shipping and accepting goods—ensured compliance with CISG principles. However, the actual issuance and validation of an L/C still had to occur outside the blockchain, relying on traditional banking institutions and external legal frameworks. The smart contract can confirm whether the L/C has been approved, allowing the transaction to proceed, but it does not autonomously handle the creation, negotiation, or enforcement of the L/C terms. This highlights a key limitation: despite automation within the blockchain environment, external financial and legal oversight remains necessary to facilitate secure international transactions, unless payment is held directly in the smart contract. In addition, execution errors due to user missteps, such as attempting actions out of sequence or misconfiguring inputs, highlighted the importance of a clear procedural understanding. The contract's design required users to follow predefined steps, emphasizing the necessity of user education in smart contract adoption, and highlighting risks associated with potential abuse.

Furthermore, the findings indicated that AI-driven contract automation enhances legal certainty and reduces transactional risks by embedding legal compliance within self-executing contracts. The enforcement of penalties for late delivery and the structured approach to contract termination provided greater assurance to both buyers and sellers. However, while the contract could be modified to allow arbitration even after contract completion, arbitration itself would still rely on external legal procedures and a panel of human arbitrators outside the smart contract. Since the contract cannot issue a legal ruling, this presents a key limitation, reinforcing the necessity of timely legal intervention before contract completion. This underscores that AI should complement, rather than replace, legal professionals by streamlining routine tasks while preserving human oversight for dispute resolution and contractual negotiations.

G Conclusion

As blockchain technologies continue to reshape international trade by improving transparency and traceability in areas such as rules of origin and customs,²⁵ their integration into legal and contractual processes is becoming increasingly viable. This research has demonstrated the potential of AI-powered smart contracts to

25 Mabrouk, H. (2024). *The urgent call for harmonizing preferential rules of origin*, p. 139. São Paulo, Brazil: Editora Dialética. <https://doi.org/10.48021/978-65-270-2768-3>.

bridge the gap between legal standards and blockchain technology, particularly within the framework of the CISG. By automating key contractual functions such as payment verification, delivery tracking, and dispute management, AI significantly enhances efficiency in contract execution. The successful deployment and verification of the CISG-compliant smart contract on the Ethereum blockchain further validated the feasibility of embedding legal principles into self-executing agreements, reducing the risk of ambiguity and misinterpretation in international trade.

However, the study also revealed that AI, despite its ability to streamline legal automation, cannot fully replace the expertise of legal professionals and, in some cases, financial institutions, as seen in the handling of Letters of Credit. The execution of smart contracts depends on precise inputs and adherence to predefined steps, which can lead to transaction errors if users do not follow the correct sequence. More critically, AI cannot resolve subjective or complex legal disputes that require human judgment. Although smart contracts can enforce penalties and automate certain contractual obligations, arbitration, and legal rulings must still be carried out by external legal bodies, as smart contracts themselves, at least for now, cannot render interpretative legal decisions.

Future research should explore ways to enhance AI's ability to handle complex legal reasoning and subjective dispute resolution within smart contracts. One potential area of improvement is the integration of AI-driven natural language processing models capable of interpreting contractual nuances and regulatory changes in real-time. Additionally, research should investigate the feasibility of hybrid legal-tech models, where smart contracts seamlessly interact with external arbitration mechanisms through blockchain-based dispute resolution protocols. Another promising avenue is the expansion of AI's role in automating financial instruments such as Letters of Credit, reducing the dependency on traditional financial intermediaries. These advancements would not only improve smart contract functionality but also strengthen the bridge between blockchain technology and international legal frameworks, paving the way for greater trust and adoption in global trade.

These findings suggest that AI should be viewed as a complementary tool rather than a substitute for legal professionals. AI can assist in automating routine contractual processes, enhancing transparency, and ensuring compliance with established legal standards. However, human oversight remains crucial for handling contractual disputes, regulatory compliance, and modifications that require contextual understanding. Moving forward, the most effective approach lies in a hybrid model where AI-powered smart contracts provide efficiency, while legal professionals ensure the robustness and adaptability of contractual agreements in dynamic trade environments.