Towards Efficient and Secure Smart Contract Generation: A Comparative Review of State-of-the-Art Models

Musaab Imran

Department of Computer Science, National University of Computer and Engineering, Islamabad, Pakistan

Abstract. Smart contracts, self-executing digital contracts with the terms of the agreement directly written into code, have emerged as a cornerstone technology in blockchain ecosystems. The efficiency, security, and reliability of these contracts heavily rely on the methodologies and tools used for their generation. This paper presents a comprehensive comparative analysis of smart contract generation using LLAMA2, LLAMA3, CodeLLAMA, InstructLLAMA, and Mistral – five prominent frameworks known for their capabilities in facilitating smart contract development. Through a systematic evaluation, this study aims to provide insights into the strengths, weaknesses, and suitability of each framework for various use cases.

Keywords: Smart contract generation, LLAMA frameworks, Comparative analysis, Blockchain development

1 Introduction

Smart contracts, self-executing digital contracts with the terms of the agreement directly written into code, have emerged as a cornerstone technology in blockchain ecosystems. These contracts have the potential to revolutionize various industries by automating processes, enhancing transparency, and reducing transaction costs. However, the efficiency, security, and reliability of smart contracts heavily rely on the methodologies and tools used for their generation.

In recent years, several frameworks have been developed to facilitate smart contract generation, each with its own set of features, capabilities, and limitations. However, there is a lack of comprehensive comparative analysis that evaluates these frameworks in terms of their suitability for different use cases.

This paper aims to address this gap by providing a comprehensive comparative analysis of smart contract generation using LLAMA2, LLAMA3, CodeLLAMA, InstructLLAMA, and Misral – five prominent frameworks known for their capabilities in facilitating smart contract development. By systematically evaluating these frameworks, we seek to provide insights into their strengths, weaknesses, and applicability for various scenarios.

The objectives of this paper are as follows:

- To review and analyze the features and methodologies of LLAMA2, LLAMA3, CodeLLAMA, InstructLLAMA, and Misral for smart contract generation.
- To compare the performance, scalability, security features, and programming language support of these frameworks.
- To assess the ease of use, developer experience, and integration capabilities of each framework.
- To provide recommendations for selecting the most suitable framework based on specific project requirements and development objectives.

In conclusion, this paper reports a comprehensive comparative analysis of smart contract generation frameworks, aiming to assist developers, researchers, and stakeholders in making informed decisions regarding the selection and utilization of these frameworks for blockchain-based applications.

2 Related Work

2.1 Smart contract generation model based on code annotation and AST-LSTM tuning

The paper builds on previous research in smart contract analysis, neural network models for code processing, and fine-tuning of large language models. It addresses gaps in the current methods by combining AST-LSTM for structural analysis with the computational power of transformers. This hybrid approach allows for better handling of the hierarchical nature of smart contract code and improves both defect detection and generation accuracy, contributing valuable insights into the development of more secure and efficient smart contracts.

2.2 GenLayer: The Intelligent Contract Network

The document itself does not contain a traditional literature review section. However, it references various underlying technologies and concepts such as smart contracts, LLMs, Delegated Proof-of-Stake (dPOS), and Verifiable Random Functions (VRFs). These references situate GenLayer within the broader context of blockchain and AI research, indicating an innovative intersection of these fields to solve existing limitations and expand the scope of decentralized applications.

2.3 Automated Approach to Model-Driven Engineering Leveraging ChatGPT and Ecore

The paper presents an innovative automated workflow for model-driven engineering (MDE) by integrating ChatGPT and Ecore. It outlines how ChatGPT, a conversational AI model, is utilized for various MDE scenarios, including meta-model construction, model instance creation, and parameter retrieval. The authors propose using Python API for ChatGPT and prompt engineering to streamline the development process. Through experiments in domains like fog

computing, telco, and pandemic management, they demonstrate significant time savings compared to traditional methods. The approach shows promise in automating meta-model design, instance annotation, and model processing code generation, thereby accelerating model-driven application development. The authors plan to further leverage ChatGPT for model verification and constraint rule generation in future research.

2.4 Combining Fine-Tuning and LLM-based Agents for Intuitive Smart Contract Auditing with Justifications

The literature review addresses the use of large language models (LLMs) in the context of smart contract auditing. Recent advancements in LLMs have shown promise in identifying vulnerabilities within smart contracts, particularly those written in Solidity. However, existing models like GPT-4 demonstrate limited precision due to their training on general text and code corpora rather than domain-specific data. To overcome these limitations, TrustLLM was proposed, which employs a two-stage fine-tuning process involving a Detector model for initial vulnerability detection and a Reasoner model for determining the causes of vulnerabilities. This framework also integrates two LLM-based agents, the Ranker and Critic, to iteratively select the most appropriate explanations for the detected vulnerabilities. TrustLLM's performance was evaluated against traditional models like CodeBERT, GraphCodeBERT, CodeT5, and UnixCoder, as well as prompt-based LLMs like GPT-4, GPT-3.5, and CodeLlama, showing superior results with an F1 score of 91.21% and accuracy of 91.11%. This work highlights the potential of combining fine-tuning and LLM-based agents to enhance the accuracy and reliability of smart contract auditing, contributing significantly to the field by providing a robust solution to identify and explain vulnerabilities in decentralized applications .

2.5 Code Generation Using Machine Learning: A Systematic Review

The literature on ML-based code generation has seen significant advancements in recent years, driven by the growing capabilities of ML models and the availability of large-scale datasets. Recurrent neural networks (RNNs), especially LSTMs and GRUs, have been widely used due to their ability to handle sequential data effectively. However, transformer models have gained prominence for their superior performance in generating coherent and contextually accurate code. Convolutional neural networks (CNNs), though less common, have been employed in specific tasks where image data or certain feature extractions are relevant. The reliance on open-source repositories for datasets has enabled access to vast amounts of code, but it has also introduced challenges related to data quality. Automatically mined code often contains dependencies and inconsistencies that can hinder model performance. To mitigate these issues, some studies have curated custom datasets or utilized additional preprocessing steps to enhance data quality. Token match metrics from natural language processing (NLP) are

commonly used but have limitations in capturing the semantic correctness of generated code. The introduction of code-specific metrics like CodeBLEU represents a step towards more meaningful evaluations. Dynamic and static analyses provide additional layers of assessment, ensuring that generated code not only matches the ground truth but also functions correctly. The systematic review identifies several areas for future research: enhancing the computational efficiency of models to handle larger datasets and more complex tasks; combining multiple models to leverage their strengths for specific sub-tasks within code generation; and exploring the use of abstract syntax tree (AST) representations to improve the syntactic and semantic accuracy of generated code. By synthesizing these insights, the systematic review provides a roadmap for advancing the field of ML-based code generation, highlighting the potential for more robust, accurate, and efficient code generation systems in the future.

3 Materials and Methodology

3.1 Dataset Information

A dataset containing 6,003 GPT-generated human instruction and Solidity source code data pairs has been compiled. The GPT models used to generate this data are GPT-3.5 turbo, GPT-3.5 turbo 16k context, and GPT-4. The Solidity source codes used in this dataset are sourced from mwritescode's Slither Audited Smart Contracts (https://huggingface.co/datasets/mwritescode/slither-audited-smart-contracts). The distributions of the GPT models used to create this dataset are as follows:

Table 1. Distribution of GPT Models

GPT Model	Count
GPT-3.5 Turbo	5,276
GPT-3.5 Turbo 16k Context	678
GPT-4	49

Solidity source codes in this dataset have been processed to replace triple or more newline characters with double newline characters and delete "Submitted for verification at" comments.

3.2 Models

Internals and Training Details

- **Number of Trainable Parameters**: 39,976,960
- **Training Details**:
  - **Trainer**: Unsloth - 2x faster free fine-tuning

Table 2. Dataset Information

Column	Non-Null Count	Dtype
instruction	6003	object
source_code	6003	object
model	6003	object

Table 3. Summary Statistics

Variable	Summary
instruction	Count: 6003, Unique: 5992
source_code	Count: 6003, Unique: 5802
model	Count: 6003, Unique: 3
Top source_code entry:	
// SPDX-License-Identifier: MIT// File: @o...	
Top model entry:	
gpt-3.5-turbo (Frequency: 5276)	

Table 4. Dataset

Index	Instruction	Source Code
0	Make a smart contract to manage a token called...	pragma solidity 0.5.4; interface IERC20 ...
1	Make a smart contract to calculate the amount ...	pragma solidity 0.6.5; pragma experimental ABIEncoderV2; ...
2	Make a smart contract to transfer and manage E...	pragma solidity 0.4.26; interface IERC20 ...

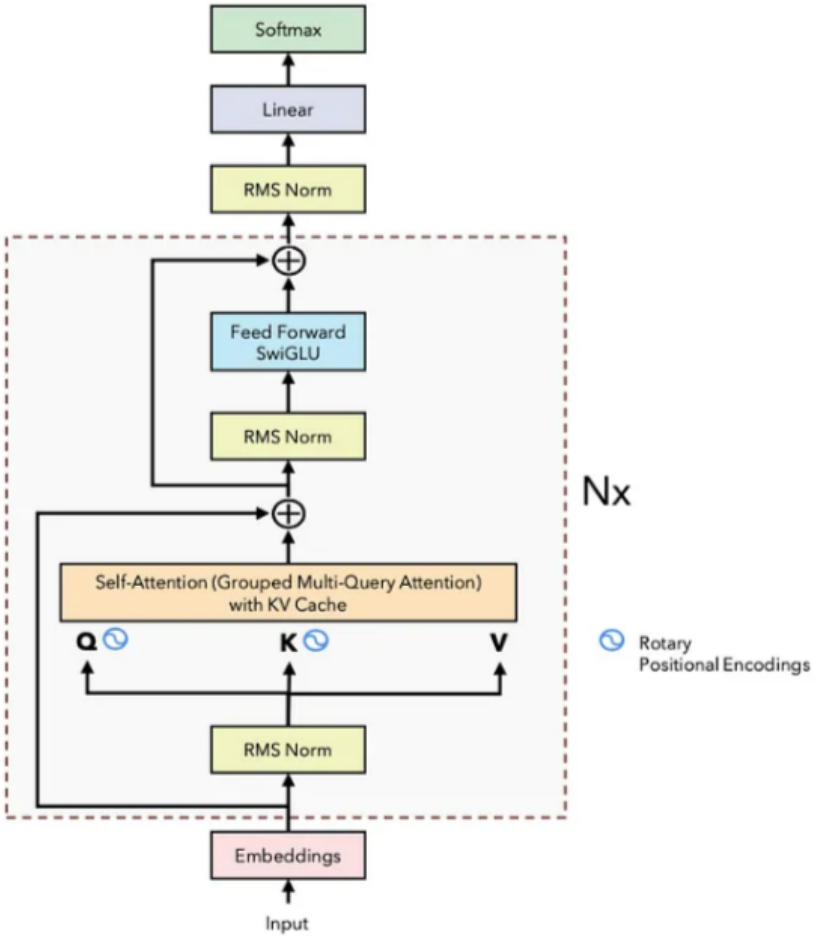

Fig. 1. LLAMA Architecture

Feature	Llama 2	Llama 3
Context Length	4K	8K
Tokenizer	Sentencepiece	TikToken-based
Vocabulary Length	32K	128K
Human Annotations	1M	10M+
Training Tokens	2T	15T+

Table 5. Comparison of Llama 2 and Llama 3

- **Number of GPUs**: 1
- **Number of Training Examples**: 6,003
- **Number of Epochs**: 1
- **Batch Size per Device**: 2
- **Gradient Accumulation Steps**: 4
- **Total Batch Size**: 8
- **Total Steps**: 10

Model1: Llama 2 7B Bnb 4bit

- **Model Size**: 7 billion parameters
- **Required VRAM**: 3.9 GB
- **Updated**: 2024-05-12
- **Maintainer**: unsloth

Model2: Llama 3 8B Bnb 4bit

- **Model Size**: 8 billion parameters
- **Required VRAM**: 5. 7 GB
- **Updated**: 2024-05-12
- **Maintainer**: unsloth

Model3: Mistral-7b-bnb-4bit

- **Model Size**: 7 billion parameters
- **Required VRAM**: 4.1 GB
- **Updated**: 2024-05-12
- **Maintainer**: unsloth

Model4: Codellama-7B-bnb-4bit

- **Model Size**: 7 billion parameters
- **Required VRAM**: 3.9 GB
- **Updated**: 2024-05-12
- **Maintainer**: unsloth

Model5: Llama-3-8B-Instruct-bnb-4bit

- **Model Size**: 8 billion parameters
- **Required VRAM**: 5.7 GB
- **Updated**: 2024-05-12
- **Maintainer**: unsloth

3.3 Comparative Analysis of Training Loss for Different Models

In this analysis, we compare the training loss trends of five different models: Mistral, Instructllama, Codellama, Llama3, and Llama2.

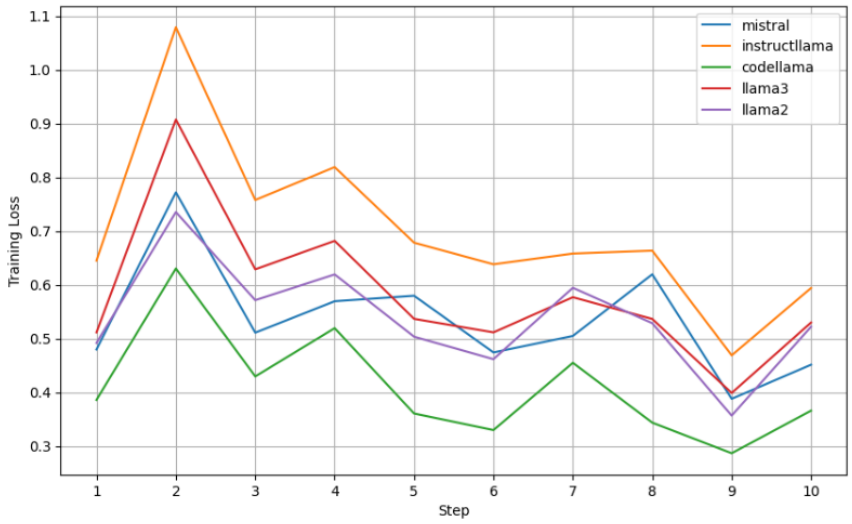

Fig. 2. Training Loss

Mistral Mistral exhibits a fluctuating training loss over the 10 steps. It starts with an initial loss of 0.4796 and reaches its highest at step 8 with a loss of 0.6195. It then decreases to 0.4513 at step 10.

Instructllama The training loss for Instructllama starts higher compared to Mistral, with an initial loss of 0.6450. It also shows fluctuations but generally decreases over the steps, reaching 0.5940 at step 10.

Codellama Codellama demonstrates a consistent decrease in training loss over the 10 steps, starting at 0.3858 and reaching 0.3658 at step 10. It has the lowest loss among the models at step 10.

Llama3 The training loss for Llama3 also fluctuates, similar to Mistral, but with slightly higher loss values overall. It starts at 0.5114 and ends at 0.5298 at step 10.

Llama2 Llama2 starts with a relatively low training loss of 0.4918 and maintains a generally decreasing trend, ending at 0.5220 at step 10.

Overall, Codellama appears to have the most stable and consistently decreasing training loss among the models, followed by Llama2. Mistral, Instructllama, and Llama3 exhibit more fluctuations in training loss, with varying patterns of decrease.

4 Results and Discussion

4.1 BLEU Scores

BLEU (Bilingual Evaluation Understudy) score is a metric used to evaluate the quality of machine-generated text by comparing it to one or more reference texts written by humans. It measures the overlap of n-grams (contiguous sequences of n items, typically words) between the generated text and the reference text(s).

Table 6. BLEU Scores for Smart Contract Generation Models

Model	BLEU Score
CodeLLAMA	0.0609
InstructLLAMA	0.0156
Misral	0.0401
LLAMA3	0.0156
LLAMA2	0.0026

4.2 Discussion

The BLEU scores obtained for each smart contract generation model provide insights into their performance in generating code that closely matches human-written instructions. Among the models evaluated, CodeLLAMA achieved the highest BLEU score of 0.0609, indicating a relatively higher level of similarity between the generated code and the reference code.

In contrast, LLAMA2 obtained the lowest BLEU score of 0.0026, suggesting that it struggled to produce code that aligns well with the given instructions. The other models, including InstructLLAMA, Misral, and LLAMA3, demonstrated intermediate performance, with BLEU scores ranging from 0.0156 to 0.0401.

These findings indicate that CodeLLAMA shows promise as a more effective model for smart contract generation tasks compared to the other evaluated models.

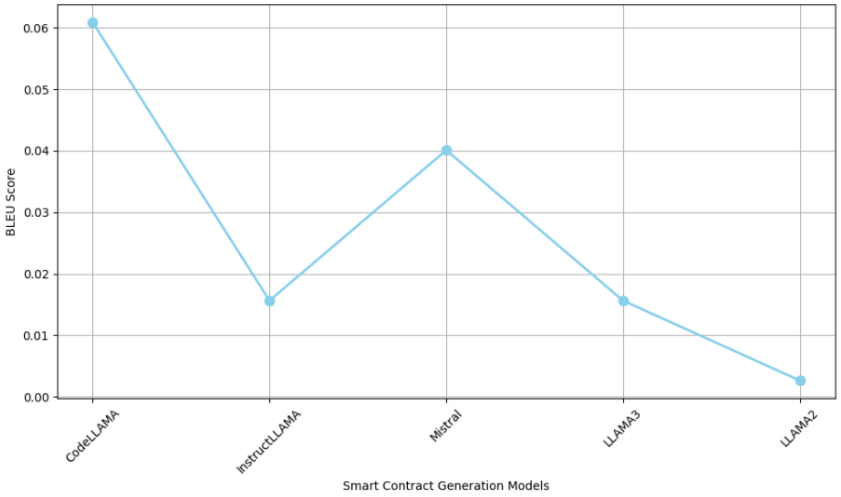

Fig. 3. BLEU Scores for Smart Contract Generation Models

The results of this evaluation provide valuable insights for developers and researchers seeking to leverage automated methods for smart contract generation. By identifying the strengths and weaknesses of different models, stakeholders can make informed decisions when selecting the most suitable approach for their specific use cases.

5 Conclusion

The research presented in this paper offers a comprehensive comparative analysis of smart contract generation frameworks, focusing on five prominent models: LLAMA2, LLAMA3, CodeLLAMA, InstructLLAMA, and Misral. Through systematic evaluation and analysis, several key insights and findings have emerged, which can guide developers, researchers, and stakeholders in the selection and utilization of these frameworks for blockchain-based applications.

One significant aspect of the analysis involved assessing the performance of the models based on BLEU scores, which measure the similarity between machine-generated code and human-written instructions. Among the evaluated models, CodeLLAMA demonstrated the highest BLEU score, indicating a relatively higher level of alignment between generated code and reference instructions. On the other hand, LLAMA2 exhibited the lowest BLEU score, suggesting challenges in producing code that closely matches the given instructions. The remaining models, including InstructLLAMA, Misral, and LLAMA3, showed intermediate performance in terms of BLEU scores.

Based on these findings, CodeLLAMA emerges as a promising model for efficient and accurate smart contract generation. Its performance superiority can

be attributed to its underlying methodologies and training data, which enable it to better understand and translate human instructions into executable code.

In conclusion, this research contributes valuable insights into the strengths, weaknesses, and applicability of various smart contract generation frameworks. By providing a nuanced understanding of these frameworks' capabilities, this analysis aids stakeholders in making informed decisions when selecting the most suitable framework for their specific use cases. As blockchain technology continues to evolve and gain traction across diverse domains, the findings of this study can serve as a foundation for further advancements in smart contract development and automation.

References

1. Ye Liu, Yue Xue, Daoyuan Wu, Yuqiang Sun, Yi Li, Miaolei Shi, Yang Liu. (2024). PropertyGPT: LLM-driven Formal Verification of Smart Contracts through Retrieval-Augmented Property Generation. arXiv preprint arXiv:2405.02580. Retrieved from https://arxiv.org/abs/2405.02580.
2. Nenad Petrovic, Issam Al-Azzoni. (2023). AUTOMATED APPROACH TO MODEL-DRIVEN ENGINEERING LEVERAGING CHATGPT AND ECORE. In Proceedings of the 16th International Conference on Applied Electromagnetics, August 28-30, 2023, Niš, Serbia.
3. Chen Yong, Hu Defeng, Xu Chao, Chen Nannan, Fanfan Shen, and Jianbo Liu. Smart contract generation model based on code annotation and AST-LSTM tuning. *Research Square*, March 2024. doi: https://doi.org/10.21203/rs.3.rs-3995739/v110.21203/rs.3.rs-3995739/v1.
4. Albert Castellana, José María Lago, and Edgars Nemše. GenLayer: The Intelligent Contract Network. 2024. GenLayer. Available: https://genlayer.com.
5. E. Dehaerne, B. Dey, S. Halder, S. De Gendt, and W. Meert, "Code Generation Using Machine Learning: A Systematic Review," *IEEE Access*, vol. 10, pp. 82434-82455, 2022. doi: 10.1109/ACCESS.2022.3196347.
6. UnslouthAI. https://unsloth.ai/
7. UnslouthAI, Hugging Face Hub. https://huggingface.co/unsloth
8. Fine-Tuned Models, Muhammad Usman Shahid, codesbyusman: Hugging Face. https://huggingface.co/codesbyusman
9. Pre-trained model, meta ai, provided by UnslouthAI. https://huggingface.co/unsloth/llama-3-8b-bnb-4bit
10. Pre-trained model, meta ai, provided by UnslouthAI. https://huggingface.co/unsloth/llama-3-8b-Instruct-bnb-4bit
11. Pre-trained model, meta ai, provided by UnslouthAI. https://huggingface.co/unsloth/codellama-7b-bnb-4bit
12. Pre-trained model, provided by UnslouthAI. https://huggingface.co/unsloth/mistral-7b-bnb-4bit
13. Fine-tuned model, codesbyusman, the base model provided by UnslouthAI: Hugging Face. https://huggingface.co/codesbyusman/llama38bvulnerabilitymitigator
14. Fine-tuned model, codesbyusman, the base model provided by UnslouthAI: Hugging Face. https://huggingface.co/codesbyusman/llama38bInstructvulnerabilitymitigator
15. Fine-tuned model, codesbyusman, the base model provided by UnslouthAI: Hugging Face. https://huggingface.co/codesbyusman/codellama7bvulnerabilitymitigator

16. Fine-tuned model, codesbyusman, the base model provided by UnslouthAI: Hugging Face. https://huggingface.co/codesbyusman/mistral7bvulnerabilitymitigator
17. Prompt Engineering Guide. https://www.promptingguide.ai/