Automated Smart Contract Vulnerability Detection using Fine-tuned Large Language Models

Zhiju Yang

Seattle University, zhijuyang@seattleu.edu

Gaoyuan Man

Arizona State University, gman2@asu.edu

Songqing Yue

University of Wisconsin - Platteville, yues@uwplatt.edu

As decentralized finance (DeFi) built on blockchain grows rapidly, the security of smart contracts underpinning DeFi has become a major concern due to exploits leading to billions in damages. Although tools exist for automated vulnerability detection in smart contracts, studies show that most vulnerabilities remain undetected. In this work, we propose using fine-tuned large language models (LLMs) for enhanced automated detection of vulnerabilities in smart contracts. We collected over 26,727 labeled smart contract vulnerabilities and fine-tuned the 13B parameter Llama-2 model. Evaluation of 1,000 unseen functions shows promising precision of 31-36% in predicting vulnerability categories. The fine-tuned LLM demonstrates potential as an auxiliary tool to identify vulnerable code and assist auditors. Future work is outlined for improving performance via larger models, higher-quality data, and specialized binary detection models. We present promising preliminary results on integrating LLMs into smart contract analysis and motivate further research at the intersection of LLMs and blockchain security.

CCS CONCEPTS • Security and privacy; Software and application security; Software security engineering

Additional Keywords and Phrases: Vulnerability detection, Smart contract, Security, Large language model

ACM Reference Format:

Zhiju Yang, Gaoyuan Man, and Songqing Yue. 2023. Automated Smart Contract Vulnerability Detection using Fine-tuned Large Language Models. In ICBTA'23: 6th International Conference on Blockchain Technology and Applications, December 15–17, 2023, Xi'an, China. ACM, Xi'an, China, 9 pages.

1 INTRODUCTION

Smart contracts have arisen as the foundational elements of decentralized finance (DeFi), offering a programmable and automated mechanism for the execution of financial transactions. Nevertheless, the security of these smart contracts has garnered substantial attention due to a multitude of security hacks, such as flash loan [2], reentrancy [3], and frontrunning [4], which have led to billions of USD in damages [1]. Even though there exist various analysis tools [6] for smart contract vulnerability detection, recent research has shown that around 80% of these vulnerabilities remain undetected by existing tools [5].

The year 2023 has witnessed a remarkable surge in the proliferation of Large Language Models (LLMs). Models such as ChatGPT [1], GPT-4 [33], Llama 2 [9], and Claude 2 [34] have demonstrated impressive performance across a wide range of neutral language processing tasks, implicating the great potential to transform our daily lives [35]. More specifically, a variety of fine-tuned LLMs have been adapted to specific downstream tasks. In closed question-answering, these LLMs can provide precise answers based on the provided context; in generation tasks, these models can produce human-like text, such as story writing or

poem composition. The adaptability and versatility of LLMs make them invaluable tools for a wide range of applications, empowering businesses, researchers, and individuals to accomplish diverse tasks with extraordinary efficiency and accuracy.

In light of the numerous successful tasks of using fine-tuned LLMs, we set out to fine-tune an LLM capable of detecting potential security issues in a given smart contract function. We first collected 3,778 security audit reports that consisted of the details of 26,727 security issues. Then, we selected Meta's Llama 2[1] with a model size of 13 billion parameters [9] as the base model for fine-tuning. As to the fine-tuning approach, we use Supervised Fine-Tuning (SFT) [10], under which the model is fine-tuned on a dataset of instructions and responses. During the SFT process, the model's weights will be adjusted to minimize the difference between the generated answers and ground-truth responses, acting as labels. In our experimental setup, we crafted prompt instructions consisting of input functions and their expected responses, representing the category of security issues associated with the given smart contract function. Fine-tuning LLMs is often prohibitively costly. Therefore, we adopted the state-of-the-art Parameter-Efficient Fine-Tuning (PEFT) method [11], which optimizes efficiency while preserving performance comparable to that achieved through full fine-tuning. PEFT methods only fine-tune a small number of extra model parameters, thereby greatly decreasing the computational and storage costs.

In our evaluation experiments, we assessed the fine-tuned model by feeding 1,000 functions and prompting the model to predict the corresponding category among the nine predefined ones. The outcomes included 78 true positives (TP), 262 true negatives (TN), 140 false positives (FP), and 520 false negatives (FN). Consequently, the model exhibited an accuracy of 34% and a precision of 36%. In contrast, the original Llama-13B unexpectedly verdicts logical issues for the majority (over 90%) of the given functions, indicating the necessity of fine-tuning to tailor the model for our specific task. In a similar context, David et al. [36] evaluated the capability of the original GPT-4 and Claude-v.13 in discovering smart contract vulnerabilities and obtained a precision of 4%. Taken together, the fine-tuned LLM has shown its great potential in discovering security issues in the given smart contract functions.

To summarize, our key contributions are:

- Compiling a dataset of 26,727 labeled Solidity functions with known vulnerabilities, plus 20,000 benign functions
- Conducting supervised fine-tuning of Llama2-13B and CodeLlama on this dataset to specialize in smart contract vulnerability detection
- Demonstrating the potential of fine-tuned LLM with a precision of 36% on unseen data
- Proposing future work such as larger models, higher-quality data, and binary classification

The fine-tuned LLM shows potential as an auxiliary tool to identify vulnerable code and assist auditors. We present initial promising results on integrating LLMs into smart contract analysis and outline directions for improvement and motivate further research at the intersection of LLMs and blockchain security.

2 RELATED WORK

Smart Contract Vulnerability Detection. Towards securing blockchain applications and smart contracts, numerous tools and methodologies have been proposed to detect vulnerabilities within the smart contract

[1] Llama 2 and its family models are available free of charge for research and commercial use.

ecosystem. Researchers have systematically surveyed and summarized the security and privacy challenges on different blockchains and smart contracts [1, 11, 12, 13, 14, 27].

In the space of smart contract analysis, several automated vulnerability detection tools have been proposed. Tsankov et al. [15] proposed Securify which is a scalable and fully automated security analyzer of Ethereum smart contracts. It symbolically analyzes the dependency graph of the given smart contract and identifies violation patterns for security hole detection. Tikhomirov et al. [16] proposed SmartCheck - an extensible static analysis tool that translates smart contract source code into an XML-based representation for pattern recognition. To facilitate and ease the automated analysis, testing, and debugging using static analysis tools, Ferreira et al. [17] proposed an easy-to-use execution framework called SmartBugs in 2020. Furthermore, Feist et al. [6] proposed Slither, which runs a suite of more than 70 vulnerability detectors. Slither is still actively developed and is widely used as of August 2023. Alongside these automated tools, a wide range of static analysis approaches have also been investigated, for example, fuzz testing [18, 19], symbolic execution [20, 21, 22], model checking [23], and formal verification [24, 25, 26].

Large Language Models. The year 2023 has witnessed a remarkable surge in the proliferation of Large Language Models (LLMs). Models such as ChatGPT [1], GPT-4 [33], Llama 2 [9], and Claude 2 [34] have demonstrated impressive performance across a wide range of neutral language processing tasks, implicating the great potential to transform our daily lives [35]. While existing LLMs have demonstrated remarkable capabilities in traditional natural language processing tasks, they may still exhibit limitations in terms of domain-specific knowledge required for particular applications, such as programming, medicine, law, and finance. Instruction tuning emerges as an effective strategy for customizing existing general-purpose LLMs into domain-specific experts. For example, Codex [37], introduced by OpenAI in July 2021, represents a GPT-based model that is fine-tuned using an extensive GitHub code corpus. This endeavor showcased Codex's remarkable ability to solve complex programming challenges. By fine-tuning the base model Flan-PaLM [38] using medical datasets, the resulting Med-PaLM [39] model functions as a medical knowledge assistant and achieves performance levels on par with expert clinicians.

In this work, we chose Llama models for fine-tuning. The collection of Llama models [9, 40, 41] was introduced by Meta AI in 2023. Since their unveiling, Llama has gained significant attention from both the research and industry communities and has delivered exceptional performance across a spectrum of open benchmarks. Notably, instruction tuning applied to Llama has emerged as a prominent method for crafting tailored or specialized models, primarily due to its relatively modest computational requirements. For example, Code Llama [41], fine-tuned based on Llama 2, is state-of-the-art for publicly available LLMs on coding tasks such as code completion and generation.

3 Data Collection and Dataset

In this work, we adopt supervised fine-tuning to adapt a pretrained LLM to the task of smart contract vulnerability detection. During supervised fine-tuning, the pre-trained LLM is trained further on a dataset of labeled vulnerabilities to adjust its weights. This allows the LLM to learn smart contract vulnerability patterns and nuances to specialize in vulnerability detection. Since there is no readily available dataset specifically dedicated to smart contract vulnerabilities, we initiate our work by assembling a dataset comprising smart contract code samples along with associated vulnerabilities. This dataset serves as the foundation upon which we conduct supervised fine-tuning to empower the LLM for effective vulnerability detection.

Nowadays, smart contract auditing companies thoroughly analyze and test smart contract code to identify vulnerabilities, errors, and ways to optimize the contracts before deployment. The audit report provided by auditors summarizes the findings and recommendations from the entire auditing process. Indeed, the audit reports alongside the corresponding smart contract code serve as ideal resources for the process of supervised fine-tuning in our research. These reports and code samples provide valuable labeled data, facilitating the refinement of our model's capabilities to detect vulnerabilities in smart contracts effectively.

Table 1: Total number of functions in each of the eight vulnerability categories and their definitions

Category	Count	Definition
gas-optimization	2,861	A gas optimization opportunity refers to the code that does not affect the functionality but can be optimized to generate different, more optimal EVM opcodes, resulting in a lower gas cost of a transaction.
coding-style	3,888	A coding style issue refers to the code where coding practices can be improved to make the code more understandable and maintainable, though the code itself may not be vulnerable.
logical	8,075	A logical issue refers to a "logical error" or "semantic error" that occurs when the code compiles and runs without any syntax errors, but the program does not behave as expected. In other words, the code is logically flawed, causing unintended or incorrect outcomes.
privilege	7,070	A privilege issue in smart contract code refers to a situation where certain users or entities gain unauthorized or excessive privileges or access to functions, data, or capabilities within a blockchain-based smart contract.
volatile-code	3,939	A volatile code issue refers to code that is unstable or unpredictable in its behavior and may produce unexpected results, pose security risks, or lead to unreliable outcomes.
inconsistency	514	An inconsistency issue refers to a situation where there is a lack of alignment or coherence between different parts of the code or between the contract state and its intended logic.
math-operations	452	A math operation issue refers to incorrect or insecure mathematical operations that could lead to overflow, underflow, loss of precision, etc.
language-specific	328	A language-specific issue refers to a problem or vulnerability that arises due to the specific features, syntax, or behavior of the Solidity programming language.
Total	26,727	

As per the compiled provider list [31], the landscape of smart contract auditing services is currently comprised of nearly one hundred companies and organizations. Notably, among this extensive array of auditing providers, CertiK [30] emerges as a preeminent blockchain security company, boasting an impressive track record of auditing over 4,000 projects since its establishment in 2018. The audit reports released by CertiK contain the description of any identified vulnerabilities, corresponding severity, and suggested remediations. Most reports are publicly available for the purpose of project marketing and user confidence building. Moreover, CertiK shares and maintains all the project details and audit reports on its Security Leaderboard webpage[2]. To the best of our knowledge, CertiK stands as the leader in terms of the sheer volume of projects audited, establishing a considerable lead over its competitors in the field. Given its dominant position within the realm of blockchain security auditing, we have opted to select CertiK as our primary data source to collect smart contract code and the corresponding vulnerabilities.

In total, we collected 3,778 audit reports ranging from June 2018 to August 2023. After filtering out penetration testing reports, duplicate audit reports, and non-Solidity projects, we compiled a set of 26,727

[2] The security leaderboard can be accessed at https://www.certik.com

Solidity functions and their vulnerability categories, as summarized in Table 1. Besides, we also randomly sampled 20K Solidity functions that have no security issues. Taken together, we compiled a dataset of 46,727 labeled functions, where each label is one of the nine categories (i.e., the eight vulnerability categories and one "none of them" for benign functions). We further partition this dataset into two subsets: 80% (i.e., 37,381) for fine-tuning and 20% (i.e., 9,345) for testing. This partition allows us to fine-tune the base LLM on a substantial portion of the data while retaining a separate, unseen portion for assessing its performance.

4 Fine-tuning Setup

In this work, we selected Llama-2-13b-chat-hf[3] and CodeLlama-13b-Instruct-hf[4] from HuggingFace as the base models for fine-tuning. We leveraged Autotrain [29] to automate the fine-tuning process using our compiled dataset. AutoTrain, developed by HuggingFace, is a tool that allows users to train and deploy state-of-the-art machine learning models for various tasks including LLM fine-tuning without writing any code. Since fine-tuning LLMs is often prohibitively costly. Therefore, we adopted the state-of-the-art Parameter-Efficient Fine-Tuning (PEFT) method [28], which optimizes efficiency while preserving performance comparable to that achieved through full fine-tuning. PEFT methods only fine-tune a small number of extra model parameters, thereby greatly decreasing the computational and storage costs. Table 2 presents an example of the fine-tuning data used in the supervised fine-tuning process.

Table 2: The prompt example used in the supervised fine-tuning process.

```
Below is an instruction that describes a task, paired with an input that provides further context.
Write a response that appropriately completes the request.

### Instruction:
You are the best smart contract auditor and Solidity expert in the world who excels at finding
vulnerabilities and optimization opportunities in Solidity code. Review the given code in the input
section in detail and very thoroughly. Think step by step very carefully. Here are the definitions
of each issue: [truncated definitions of each of the eight vulnerabilities as described in Table
1]. Now, you need to answer this question: what kind of issue does the following code have? Choose
your answer from this list: gas-optimization, coding-style, logical, privilege, volatile-code,
inconsistency, math-operations, language-specific, none of them.

### Input:
{an input smart contract function}

### Response:
{the anticipated vulnerability category in the given list}
```

We used two GeForce RTX 3090 (each has 24GB of memory) GPUs to fine-tune the selected base models. The fine-tuning process takes 35GB of GPU memory with 16-bit floating point numbers (i.e., half

[3] This model can be accessed at https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
[4] This model can be accessed at https://huggingface.co/codellama/CodeLlama-13b-Instruct-hf

precision) enabled. The whole process involved three epochs and each epoch takes roughly 15 hours. In the following sections, the two fine-tuned models are referred to as FT-Llama-2-13B and FT-CodeLlama-13B, respectively.

5 Overall Performance

To evaluate the fine-tuned models, denoted as FT-Llama2-13B and FT-CodeLlama-13B, respectively, we randomly selected 1,000 smart contract functions from the test subset of our dataset, which have never been seen during the fine-tuning process. Table 3 provides an overview of the vulnerability distribution of these 1,000 testing functions.

Table 3: The vulnerability distribution of the 1,000 testing functions.

Category	Percentage	Category	Percentage	Category	Percentage
gas-optimization	6.3%	volatile-code	9.3%	privilege	15%
coding-style	9%	inconsistency	1%	none	40.2%
logical	16.7%	math-operations	0.8%	language-specific	1.8%

Table 4 summarizes the performance of the fine-tuned models in predicting the vulnerability category of these testing functions. Overall, FT-CodeLlama-13B achieves an accuracy of 34%, accompanied by precision and recall rates of 36% and 13%, respectively, while FT-Llama2-13B has a slightly worse performance. In comparison, the base Llama2-13B model demonstrated a binary classification tendency, wherein it categorized over 90% of the tested functions as having logical issues, while categorizing the remainder as having no issues. This indicates that the base model has inherent limitations (considering the data used for training) when it comes to performing vulnerability detection tasks without the process of fine-tuning. In a similar context where researchers used GPT-4 and Claude1.3 to detect vulnerabilities in 52 smart contracts, the models achieved a precision of only 4% [36].

Table 4: The performance of the fine-tuned models in predicting the vulnerability category of 1,000 testing functions.

Fine-tuned Model	TP	TN	FP	FN	Accuracy	Precision	Recall
FT-CodeLlama-13B	78	262	140	520	34%	36%	13%
FT-Llama2-13B	76	233	169	522	31%	31%	13%

Despite the relatively low recall of the fine-tuned models, they show promise in assisting smart contract auditors in identifying potentially vulnerable code segments. In an auditing context, these models can be employed as valuable vulnerability alert tools, especially considering their precision rate of 36%. This signifies that approximately one out of every three alerts generated by these models will indeed correspond to valid vulnerabilities. While the recall rate may indicate that some vulnerabilities could be missed, the models still offer a valuable resource for auditors by efficiently pinpointing areas of concern within smart contract codebases, ultimately enhancing the auditing process and reducing the risk of undetected vulnerabilities.

6 Discussion

Indeed, we consider this work to be just the beginning of integrating fine-tuned large language models into vulnerability detection and smart contract auditing. The promising results and insights gained from this work

lay the foundation for further advancements and refinements in this evolving field. In particular, we envision the following avenues for improvement.

First, the potential for enhancing vulnerability detection performance can be explored by fine-tuning larger language models. Our preliminary experiments with Llama2-7B revealed limitations in generating the categorical responses we aimed for in our context. Consequently, our adoption of models with 13 billion parameters has proven to be a promising starting point. Looking ahead, the fine-tuning of even more substantial models, such as Llama2-70B or OpenAI GPT-3.5, presents an opportunity to achieve even higher levels of performance and precision in the task of vulnerability detection.

Second, we recognize the pivotal role of data quality in further enhancing model performance. As part of our future endeavors, we plan to undertake a meticulous manual selection process to curate high-quality data, prioritizing instances of high-severity vulnerabilities. This approach will ensure that our models are exposed to the most critical and relevant information, thereby improving their capability to identify and address significant vulnerabilities effectively.

Third, we acknowledge the potential benefits of specialized binary detection models that concentrate on specific categories of vulnerabilities. For instance, dedicating the fine-tuning process to the detection of a particular vulnerability type, such as reentrancy issues, holds promise in achieving heightened precision and recall for that specific class of vulnerabilities. This targeted approach can cater to the unique characteristics and challenges posed by distinct vulnerability categories, further improving the effectiveness of our vulnerability detection solutions.

7 Conclusion

In this work, we have demonstrated the potential of using fine-tuned LLMs for smart contract vulnerability detection. Our experiments involved fine-tuning Llama-2-13B and CodeLlama-13B models on a dataset of over 40,000 labeled Solidity functions. Our evaluation showed promising precision rates of 31-36% in predicting vulnerability categories on unseen functions. This validates the ability of fine-tuned LLMs to learn nuances of smart contract vulnerabilities and serve as an auxiliary tool to alert auditors. Moving forward, we have outlined promising directions like utilizing larger models, curating higher-quality training data, and developing specialized models. As LLMs continue advancing rapidly, integrating them into smart contract analysis holds immense potential.

References

[1] Liyi Zhou, Xihan Xiong, Jens Ernstberger, Stefanos Chaliasos, Zhipeng Wang, Ye Wang, Kaihua Qin, Roger Wattenhofer, Dawn Song, and Arthur Gervais. Sok: Decentralized finance (defi) attacks. IEEE Symposium on Security and Privacy, 2023.

[2] K. Qin, L. Zhou, B. Livshits, and A. Gervais, “Attacking the defi ecosystem with flash loans for fun and profit,” in International Conference on Financi al Cryptography and Data Security. Springer, 2021, pp. 3–32.

[3] Y. Xue, M. Ma, Y. Lin, Y. Sui, J. Ye, and T. Peng, “Cross-contract static analysis for detecting practical reentrancy vulnerabilities in smart contracts,” in 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2020

[4] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, “Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability,” in 2020 IEEE Symposium on Security and Privacy (SP). IEEE, 2020.

[5] Zhang, Zhuo, Brian Zhang, Wen Xu, and Zhiqiang Lin. "Demystifying Exploitable Bugs in Smart Contracts." ICSE, 2023.

[6] Josselin Feist, Gustavo Grieco, and Alex Groce. 2019. Slither: a static analysis framework for smart contracts. In 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). IEEE, 8–15.

[7] Lexi Brent, Neville Grech, Sifis Lagouvardos, Bernhard Scholz, and Yannis Smaragdakis. 2020. Ethainter: a smart contract security analyzer for composite vulnerabilities. In Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation. 454–469.

[8] Lexi Brent, Anton Jurisevic, Michael Kong, Eric Liu, Francois Gauthier, Vincent Gramoli, Ralph Holz, and Bernhard Scholz. 2018. Vandal: A scalable security analysis framework for smart contracts. arXiv preprint arXiv:1809.03981 (2018).

[9] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288, 2023.

[10] Ali M, Son DH, Kang SH, Nam SR. An accurate CT saturation classification using a deep learning approach based on unsupervised feature extraction and supervised fine-tuning strategy. Energies. 2017 Nov 10;10(11):1830.

[11] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on Ethereum smart contracts (sok)," in International conference on principles of security and trust, pp. 164–186, Springer, 2017.

[12] R. Zhang, R. Xue, and L. Liu, "Security and privacy on blockchain," ACM Computing Surveys (CSUR), vol. 52, no. 3, pp. 1–34, 2019.

[13] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, "A survey on the security of blockchain systems," Future Generation Computer Systems, vol. 107, pp. 841–853, 2020.

[14] W. Zou, D. Lo, P. S. Kochhar, X.-B. D. Le, X. Xia, Y. Feng, Z. Chen, and B. Xu, "Smart contract development: Challenges and opportunities," IEEE Transactions on Software Engineering, vol. 47, no. 10, pp. 2084–2106, 2019.

[15] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Bünzli, and Martin Vechev. Securify: Practical security analysis of smart contracts. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18, page 67–82

[16] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "Smartcheck: Static analysis of ethereum smart contracts," in Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain, pp. 9–16, 2018.

[17] J. F. Ferreira, P. Cruz, T. Durieux, and R. Abreu, "Smartbugs: A framework to analyze solidity smart contracts," in Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering, pp. 1349–1352, 2020.

[18] C. Liu, H. Liu, Z. Cao, Z. Chen, B. Chen, and B. Roscoe, "Reguard: finding reentrancy bugs in smart contracts," in 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion), pp. 65–68, IEEE, 2018.

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

[20] J. Chang, B. Gao, H. Xiao, J. Sun, Y. Cai, and Z. Yang, "scompile: Critical path identification and analysis for smart contracts," in International Conference on Formal Engineering Methods, pp. 286–304, Springer, 2019.

[21] J. Krupp and C. Rossow, "teEther: Gnawing at Ethereum to automatically exploit smart contracts," in 27th USENIX Security Symposium (USENIX Security 18), pp. 1317–1333, 2018.

[22] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, pp. 254–269, 2016.

[23] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "Zeus: analyzing safety of smart contracts.," in Ndss, pp. 1–12, 2018.

[24] E. Hildenbrandt, M. Saxena, N. Rodrigues, X. Zhu, P. Daian, D. Guth, B. Moore, D. Park, Y. Zhang, A. Stefanescu, et al., "Kevm: A complete formal semantics of the ethereum virtual machine," in 2018 IEEE 31st Computer Security Foundations Symposium (CSF), pp. 204–217, IEEE, 2018.

[25] D. Park, Y. Zhang, M. Saxena, P. Daian, and G. Ros ̧u, "A formal verification tool for Ethereum vm bytecode," in Proceedings of the 2018 26th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering, pp. 912–915, 2018.

[26] I. Grishchenko, M. Maffei, and C. Schneidewind, "Foundations and tools for the static analysis of Ethereum smart contracts," in International Conference on Computer Aided Verification, pp. 51–78, Springer, 2018.

[27] Yang, Zhiju, Gaoyuan Man, and Songqing Yue. "Understanding Security Audits on Blockchain." Proceedings of the 2022 5th International Conference on Blockchain Technology and Applications. 2022.

[28] Mangrulkar SO, Gugger S, Debut L, Belkada Y, Paul S. PEFT: State-of-the-art parameter-efficient fine-tuning methods.

[29] AutoTrain Advanced, 2023. https://github.com/huggingface/autotrain-advanced

[30] CertiK Security Audit, 2023. https://www.certik.com/products/security-audit.

[31] Discover the World's Top Smart Contract Auditing Companies, 2023. https://smartcontractaudits.com/audit-providers/companies/1.

[32] ChatGPT, 2023. https://openai.com/blog/chatgpt.

[33] OpenAI. GPT-4 Technical Report. arXiv preprint arXiv:2303.08774, 2023.

[34] Claude 2, 2023, https://www.anthropic.com/index/claude-2

[35] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. arXiv preprint arXiv:2303.18223, 2023.

[36] Isaac David, Liyi Zhou, Kaihua Qin, Dawn Song, Lorenzo Cavallaro, and Arthur Gervais. Do you still need a manual smart contract audit? arXiv preprint arXiv:2306.12338, 2023.

[37] Chen M, Tworek J, Jun H, Yuan Q, Pinto HP, Kaplan J, Edwards H, Burda Y, Joseph N, Brockman G, Ray A. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374. 2021 Jul 7.

[38] Chung HW, Hou L, Longpre S, Zoph B, Tay Y, Fedus W, Li E, Wang X, Dehghani M, Brahma S, Webson A. Scaling instruction-finetuned language models. arXiv preprint arXiv:2210.11416. 2022 Oct 20.

[39] Singhal K, Azizi S, Tu T, Mahdavi SS, Wei J, Chung HW, Scales N, Tanwani A, Cole-Lewis H, Pfohl S, Payne P. Large language models encode clinical knowledge. arXiv preprint arXiv:2212.13138. 2022 Dec 26.

[40] Touvron H, Lavril T, Izacard G, Martinet X, Lachaux MA, Lacroix T, Rozière B, Goyal N, Hambro E, Azhar F, Rodriguez A. Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971. 2023 Feb 27.

[41] Rozière B, Gehring J, Gloeckle F, Sootla S, Gat I, Tan XE, Adi Y, Liu J, Remez T, Rapin J, Kozhevnikov A. Code llama: Open foundation models for code. arXiv preprint arXiv:2308.12950. 2023 Aug 24.