



Catalyst Report
Open-Source Vulnerability Scanner for Cardano Smart
Contracts

UnboundedMarket Team

November 7, 2024

Contents

1	Motivation	3
2	Literature & Market Review	3
2.1	Recent Research	3
2.2	Market Overview	4
3	Open-Source Vulnerability Scanner	4
3.1	Data Collection	5
3.2	Data Processing & Paring	5
3.3	Model	6
3.4	Fine-Tuning	7
3.4.1	Approach 1: Sequence-to-Sequence Fine-Tuning	7
3.4.2	Approach 2: Combined Classification and Sequence Generation . .	7
3.5	Evaluation	8
3.5.1	Perplexity	8
3.5.2	Accuracy	9
4	Project Plan	9
4.1	Milestone 1: Initial Research and Report	9
4.2	Milestone 2: Data Collection & Preprocessing	10
4.3	Milestone 3: Model and Test Suite Development	11
4.4	Milestone 4: Fine-tuning, Feedback, and Project Closeout	11
5	Discussion	12
5.1	Summary	12
5.2	Scope & Limitations	12

1 Motivation

Smart contract vulnerabilities present a serious risk to Blockchain ecosystems, such as Cardano, and can lead to significant financial losses, undermining trust in the platform. Existing auditing processes mitigate this risk but are time-consuming and costly and may overlook critical bugs, which can hinder Cardano’s adoption and growth. Recent studies have demonstrated the potential of large language models (LLMs) to identify coding errors and vulnerabilities in smart contract languages such as Solidity, which are used on Ethereum. Building on these findings, we aim to bring these capabilities to the Cardano ecosystems by fine-tuning a large language model to identify bugs in Cardano smart contracts.

While our tool is not intended to be a standalone solution for ensuring bug-free smart contracts, it provides an additional layer of security by finding potential bugs and vulnerabilities early in the development process. Smart contract audits typically involve multiple iterations; after identifying and addressing a bug or vulnerability, an additional audit is required to confirm the fix. By enabling early vulnerability detection, we hope that our tool reduces the number of audit iterations, reducing the overall cost and time of the process. Furthermore, it may identify subtle bugs that could be overlooked in manual audits. We thus hope that our tool will improve the accuracy and efficiency of smart contract audits, reducing vulnerabilities and increasing the security of smart contracts within the Cardano ecosystem.

This report represents the initial milestone of our project. We start by analyzing recent related research in Section 2.1 and other market solutions in Section 2.2. In Section 3, we present our solution, an open-source vulnerability scanner. Further in Section 4, we discuss our project plan, reviewing the milestones and timeline. Finally, Section 5 presents the discussion and Section 5.1 the summary.

2 Literature & Market Review

In this section, we provide an overview of previous research (see Section 2.1) on using LLMs for detecting bugs in code and smart contracts. We further survey existing products on the market (Section 2.2) that use large language models for identifying bugs in smart contracts.

2.1 Recent Research

Previous research has applied LLMs for automated bug detection and correction and thereby demonstrated their effectiveness in various coding environments [22, 12, 9, 5]. Additionally, LLMs have shown promise in generating test cases, increasing code robustness through automated testing [16]. These studies typically use datasets contain-

ing coding errors to train models on typical bug patterns and security vulnerabilities [13, 7, 2].

Recent work also explores LLMs for identifying vulnerabilities and bugs in smart contracts [14, 10, 15, 1]. However, no studies have focused specifically on LLM-assisted bug detection for smart contract languages like Plutus, Aiken, or OpShin, which are used within the Cardano ecosystem.

2.2 Market Overview

In addition to recent academic research, several market solutions exist for AI-assisted smart contract audits. Projects such as AuditWizard¹, ChainGPT², and 0x0³ have introduced AI-assisted tools for auditing smart contracts.

AuditWizard offers a platform that combines static code analysis, proof-of-concept testing, AI-driven threat modeling, and automated audit report generation. Its web-based interface allows users to access these features without the need for local installations, streamlining the auditing process. ChainGPT provides AI models specifically tailored to generate and audit Solidity smart contracts. Similarly, 0x0 provides an AI Smart Contract Auditor to generate reports about potential vulnerabilities and corresponding fixes in Solidity smart contracts.

However, these tools are closed-source and, therefore, lack transparency and raise privacy concerns. Moreover, there is an absence of AI-assisted auditing tools for Cardano smart contracts. This gap can be attributed to the smaller developer community on Cardano and the complexity of its underlying language, Plutus, which is based on Haskell. While Aiken has emerged as a popular and user-friendly alternative to developing smart contracts on Cardano, its adoption within the developer community is still limited. Consequently, the development of new automated auditing tools for Cardano’s ecosystem has been limited.

3 Open-Source Vulnerability Scanner

In this section, we describe our approach to developing an open-source vulnerability scanner to detect bugs in Cardano smart contracts. As shown in Figure 1, the development consists of several key components, including the collection and preprocessing of the appropriate data and the fine-tuning of the model.

¹<https://www.auditwizard.io/>

²<https://www.chaingpt.org/>

³<https://auditor.0x0.ai/>

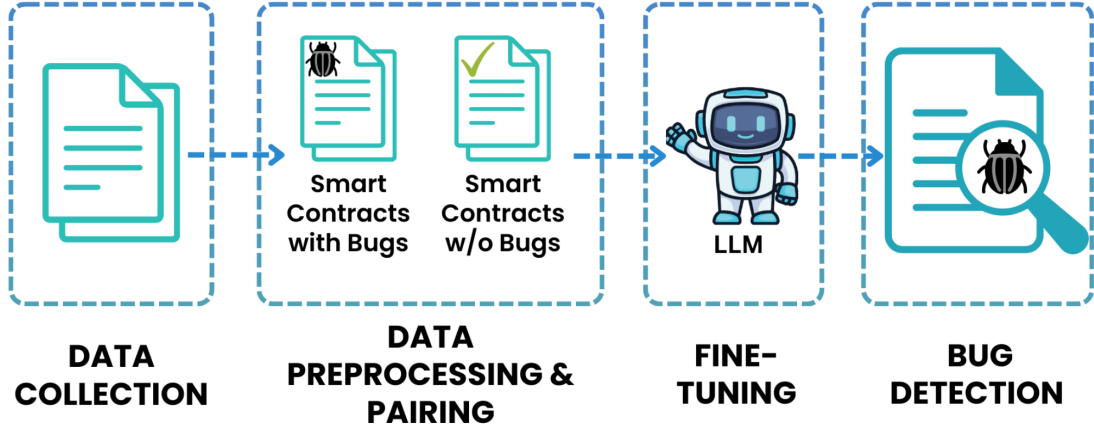


Figure 1: Overview of the individual steps for fine-tuning an LLM to detect bugs and vulnerabilities in smart contracts.

3.1 Data Collection

We start by describing the collection and preprocessing of the dataset, which we use to fine-tune our model. Ensuring high quality of the data used to train or fine-tune an AI model is important, as data quality is one of the key factors determining the performance of the resulting model [11, 3, 8].

The first step is to assemble a dataset consisting of Cardano smart contracts. To curate our dataset, we select several publicly available resources to provide various materials on the most popular smart contract languages on Cardano, namely Plutus, Aiken, and OpShin. We collect Plutus code examples from open-source repositories⁴, providing exposure to different implementations and different use cases. Additionally, we collect publicly available examples of Aiken⁶ and OpShin smart contracts⁷. Combined, these resources ensure a diverse and representative dataset of Cardano smart contracts.

3.2 Data Processing & Paring

Importantly, fine-tuning a language model on publicly available smart contracts is insufficient to teach the model how to identify bugs in smart contracts. These contracts are typically bug-free and thus lack annotated examples of common errors, which are necessary for training a model to detect vulnerabilities. Thus, we must devise a different strategy for teaching the model how to identify bugs:

⁴<https://github.com/IntersectMBO/plutus>

⁵<https://github.com/ernius/plutus-cardano-samples>

⁶<https://aiken-lang.org/installation-instructions>

⁷<https://github.com/OpShin/awesome-opshin>

We process each individual smart contract by manually introducing a range of smart contract bugs and their corresponding explanations. To introduce such bugs, we rely on common smart contract bugs across different blockchain ecosystems, as well as bugs specific to smart contract language on Cardano⁸.

This procedure leads to a dataset $\mathcal{D} = \{(c_i, e_i, y_i)\}_{i=1}^N$, which consists of N triples (c_i, e_i, y_i) , where c_i is a code snippet (either the entire smart contract a snippet extracted from a smart contract), and e_i is a string related to c_i . For buggy code snippets, e_i provides a description of the bug present in c_i . For correct code snippets, e_i is a standardized message indicating the absence of bugs, such as “No bugs found.” Each code snippet is associated with a binary label $y_i \in \{0, 1\}$, where $y_i = 1$ if c_i contains a bug and $y_i = 0$ otherwise.

Now, given a code snippet, the model is trained to predict whether it contains a bug or vulnerability and to generate a corresponding explanation. Note that this is a common approach in other areas of natural language processing, such as grammatical error correction (GEC), where grammatical errors are often synthetically created to obtain large datasets for training models [18].

3.3 Model

We leverage existing pretrained large language models and fine-tune them to detect bugs and vulnerabilities on Cardano smart contracts.

Formally, an LLM, denoted by P_θ , estimates the probability of a sequence of tokens $X = (x_1, x_2, \dots, x_T)$ by decomposing the joint probability into a product of conditional probabilities using the chain rule:

$$P_\theta(X) = \prod_{t=1}^T P_\theta(x_t \mid x_{<t}), \quad (1)$$

where $x_{<t} = (x_1, x_2, \dots, x_{t-1})$ represents the sequence of preceding tokens and θ denotes the model’s parameters.

During pretraining, the model learns the conditional probabilities by predicting the next token in a sequence, given all the previous tokens. Formally, the objective is to minimize the negative log-likelihood over a corpus of text, i.e.,

$$\mathcal{L}(\theta) = - \sum_{i=1}^N \sum_{t=1}^{T_i} \log P_\theta(x_t^{(i)} \mid x_{<t}^{(i)}), \quad (2)$$

where N is the number of sequences in the training dataset, T_i is the length of the i -th sequence and $x_t^{(i)}$ is the t -th token in the i -th sequence.

⁸See <https://library.mlabs.city/common-plutus-security-vulnerabilities> for a list of common bugs in Plutus.

To be suitable for our purpose, a model must be open-source, sufficiently large, and pretrained on large amounts of text. We thus select our model from the Llama family [19, 20, 6, 17]. These pretrained, open-source large language models, developed by Meta AI, are built upon the Transformer architecture introduced by Vaswani et al. (2017) [21].

3.4 Fine-Tuning

Given our dataset \mathcal{D} , as explained in Section 3.2 and model P_θ , as detailed in Section 3.3, we explore two distinct approaches for fine-tuning a large language model: (1) a sequence-to-sequence (seq2seq) method and (2) a combined classification and sequence generation method.

3.4.1 Approach 1: Sequence-to-Sequence Fine-Tuning

In the first approach, we model bug detection and explanation as a sequence-to-sequence task. We fine-tune the pretrained LLM to generate the corresponding explanation e_i given the input code snippet c_i . This approach mainly relies on the model’s inherent ability to understand and generate natural language based on code input acquired during pretraining.

We fine-tune the model using the standard cross-entropy loss over the output sequence:

$$\mathcal{L}_{\text{seq}} = - \sum_{t=1}^T \log P_\theta(e_i^t \mid c_i, e_i^{<t}),$$

where T is the length of the explanation e_i , e_i^t is the token at position t , $e_i^{<t}$ represents all preceding tokens, and $P_\theta(e_i^t \mid c_i, e_i^{<t})$ is the probability assigned by the model parameters θ to the token e_i^t , conditioned on the input c_i and previous tokens $e_i^{<t}$.

In this setup, we train the model using teacher forcing, where the ground truth tokens $e_i^{<t}$ are provided as input when predicting the next token e_i^t . The model learns to generate appropriate explanations for both buggy and correct code snippets based solely on the input code.

3.4.2 Approach 2: Combined Classification and Sequence Generation

Recognizing that the seq2seq approach may not be sufficient for detecting bugs in the first place, we introduce a second approach that combines classification with sequence generation. In this approach, we first determine whether a code snippet is buggy and then generate an explanation of the bug if necessary.

We extend the pre-trained model by adding a classification head to its architecture. The model processes the input code snippet c_i to generate contextual representations. The classification head consists of a feedforward neural network layer, which takes the encoded representation output and computes the probability $p_i = P_\theta(y_i = 1 \mid c_i)$ that c_i contains a bug. If $y_i = 1$, indicating that the code is buggy, the model generates the explanation e_i using the encoded representations and previous output tokens.

We define a combined loss function that integrates both classification and sequence generation objectives. The classification loss $\mathcal{L}_{\text{class}}$ is defined using binary cross-entropy:

$$\mathcal{L}_{\text{class}} = -[y_i \log p_i + (1 - y_i) \log(1 - p_i)].$$

This loss measures the error in predicting whether c_i is buggy. The sequence generation loss \mathcal{L}_{seq} is calculated as:

$$\mathcal{L}_{\text{seq}} = - \sum_{t=1}^T \log P_\theta(e_i^t \mid c_i, e_i^{<t}).$$

The total loss $\mathcal{L}_{\text{total}}$ combines both losses:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{class}} + \lambda \cdot y_i \cdot \mathcal{L}_{\text{seq}},$$

where λ is a hyperparameter that balances the importance of the classification and generation tasks. The term y_i ensures that the sequence generation loss is only considered for buggy code snippets ($y_i = 1$), preventing the model from generating bug explanations when no bugs are present.

During training, the model processes the input code snippet c_i to produce hidden representations. The classification head computes the probability p_i that c_i contains a bug. If $y_i = 1$, the decoder generates the explanation e_i using the encoded representations and previous output tokens. The total loss $\mathcal{L}_{\text{total}}$ is computed and backpropagated to update the model parameters θ .

3.5 Evaluation

We evaluate both the sequence-to-sequence approach and the combined classification and sequence generation approach using a held-out test dataset of smart contract code snippets. Our evaluation focuses on two primary metrics: perplexity and accuracy.

3.5.1 Perplexity

Perplexity is a measure of how well a language model predicts a sample. It is defined as the exponential of the average negative log-likelihood per token. For a test set of N

code snippets and their corresponding explanations, the perplexity \mathcal{P} is computed as:

$$\mathcal{P} = \exp \left(\frac{1}{\sum_{i=1}^N T_i} \sum_{i=1}^N \sum_{t=1}^{T_i} -\log P_{\theta}(e_i^t \mid c_i, e_i^{<t}) \right), \quad (3)$$

where T_i is the length of the explanation e_i for the i -th code snippet and $P_{\theta}(e_i^t \mid c_i, e_i^{<t})$ is the probability assigned by the model with parameters θ to the t -th token e_i^t in the explanation, given the code snippet c_i and all previous explanation tokens $e_i^{<t}$.

3.5.2 Accuracy

Accuracy measures the proportion of correct predictions made by the model. For the combined approach, which includes an explicit classification component, accuracy \mathcal{A} is defined as:

$$\mathcal{A} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Samples}} = \frac{TP + TN}{TP + TN + FP + FN}, \quad (4)$$

where TP (True Positives) are buggy code snippets correctly classified as buggy, TN (True Negatives) are correct code snippets correctly classified as correct, FP (False Positives) are correct code snippets incorrectly classified as buggy and FN (False Negatives) are buggy code snippets incorrectly classified as correct.

For the sequence-to-sequence model, which lacks an explicit classification layer, we consider a code snippet as classified as buggy if the model generated an explanation other than "No bugs found."

4 Project Plan

In this section, we provide a quick overview of the individual steps, according to the milestones of this project; see Figure 2.

4.1 Milestone 1: Initial Research and Report

Description: In this initial phase, we conduct the necessary foundational research, including the project objectives, relevant literature, and our general method. The outcome of this milestone is an initial report detailing the project scope and a structured data collection pipeline. This milestone is marked as completed with the completion of this report.










TIME PROCESS	MILESTONE 1	MILESTONE 2	MILESTONE 3	MILESTONE 4
Research & Initial Report				
Data Collection				
Data Preprocessing & Data Pipeline Development				
Model Pipeline Development				
Test Suite Development				
Fine-tuning & Iterative Improvements				
Release & Community Feedback				
Iterative Improvement based on Feedback				
Closeout Report and Video				

Figure 2: Roadmap for developing the open-source vulnerability scanner for Cardano smart contracts.

Timeline: The duration of this milestone is two months, and the milestone is expected to be completed at the end of October 2024.

Resource Allocation: For this milestone, two researchers are responsible for foundational research and outlining the technical approach to developing the vulnerability scanner. The primary resources are access to research databases and design tools to create the graphic material.

4.2 Milestone 2: Data Collection & Preprocessing

Description: The second milestone is centered around collecting and processing data. As detailed in Section 3.1, this step is important for the performance of our final model. After collecting various smart contracts, the raw data undergoes preprocessing to create a dataset for fine-tuning our models to detect bugs in smart contracts, as described in Section 3.2. This milestone concludes with developing a functional data pipeline to

fine-tune our language model. The deliverables for this milestone are the documentation and release of our dataset and code used for preprocessing.

Timeline: The expected duration of this milestone is two months, and the milestone is expected to be completed at the end of December 2024.

Resource Allocation: This milestone requires data engineers to focus on data collection and the setup of the data pipeline. Key resources include cloud storage for storing the curated dataset and data processing tools.

4.3 Milestone 3: Model and Test Suite Development

Description: This phase focuses on developing a scalable model pipeline that includes the training processes and evaluation metrics needed for fine-tuning. Simultaneously, we develop a test suite to benchmark our fine-tuned model's performance. The final step of this milestone is to fine-tune and test the model. The deliverables for this milestone are the documentation and release of our code, which is used to train and test the model.

Timeline: The expected duration of this milestone is two months, and the milestone is expected to be completed at the end of February 2025.

Resource Allocation: This milestone requires machine learning engineers to fine-tune and test the model. Additionally, this milestone requires access to GPU clusters and machine learning libraries to build and evaluate the model.

4.4 Milestone 4: Fine-tuning, Feedback, and Project Closeout

Description: The final phase is centered around the official release and announcements of our model. Post-release, we collect community feedback and make further refinements based on the feedback and observed real-world performance. The project concludes with a closeout report and an accompanying video documenting the development process, key findings, and future work.

Timeline: The expected duration of this milestone is one month, and the milestone is expected to be completed at the end of March 2024.

Resource Allocation: This final milestone requires machine learning engineers and a community manager. The machine learning engineers implement the final model tuning,

while the release and feedback collection will be handled by the community manager. Just like milestone 3, this milestone requires access to GPU clusters and machine learning libraries.

5 Discussion

Here, we provide a final discussion of our report, including a short summary in Section 5.1 and a section where we revisit the scope and limitation (see Section 5.2).

5.1 Summary

This report outlines our approach to developing an open-source vulnerability scanner aimed at detecting bugs in Cardano smart contracts. We began by conducting a thorough analysis of existing research and market trends (Section 2.1, Section 2.2), which revealed a gap in automated auditing tools for smart contracts within the Cardano ecosystem. In Section 3, we detailed our methodology for collecting and processing smart contracts, emphasizing the challenges associated with sourcing and curating a relevant dataset. We then proposed a strategy for fine-tuning large language models (LLMs) to detect vulnerabilities specific to Cardano’s Plutus language. While we have formulated concrete objective functions for the fine-tuning process, we acknowledge that iterative refinements may be necessary based on the model’s performance during the testing phases. The project’s roadmap is presented in Figure 2, describing the individual steps and milestones toward the development of our vulnerability scanner.

5.2 Scope & Limitations

In a recent study, David et al. 2023, [4] investigate whether code audits from LLMs could potentially replace human audits. Importantly, they conclude by acknowledging that “while LLMs augment the auditing process, they do not replace the need for human auditors at this stage.” We thus re-emphasize that the goal of this project is not to develop a single end-to-end auditing tool. Instead, we develop a tool to assist developers and auditors in identifying bugs in smart contracts by integrating the vulnerability scanner into a human-in-the-loop setting. We hope that this improves the efficiency and security of smart contract development on Cardano.

References

- [1] Md Tauseef Alam, Raju Halder, and Abyayananda Maiti. Detection made easy: Potentials of large language models for solidity vulnerabilities, 2024.
- [2] Guru Bhandari, Amara Naseer, and Leon Moonen. Cvefixes: automated collection of vulnerabilities and their fixes from open-source software. In *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering*, PROMISE 2021, page 30–39, New York, NY, USA, 2021. Association for Computing Machinery.
- [3] Lukas Budach, Moritz Feuerpfeil, Nina Ihde, Andrea Nathansen, Nele Noack, Hendrik Patzlaff, Felix Naumann, and Hazar Harmouch. The effects of data quality on machine learning performance, 2022.
- [4] Isaac David, Liyi Zhou, Kaihua Qin, Dawn Song, Lorenzo Cavallaro, and Arthur Gervais. Do you still need a manual smart contract audit?, 2023.
- [5] David de Fitero-Dominguez, Eva Garcia-Lopez, Antonio Garcia-Cabot, and Jose-Javier Martinez-Herraiz. Enhanced automated code vulnerability repair using large language models. *Engineering Applications of Artificial Intelligence*, 138:109291, 2024.
- [6] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plaw-iak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Lauren Rantala-Yearly, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan,

Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collet, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenxin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuwei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aaron Grattafiori, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkan Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesco Caggioni, Francisco Guzmán, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry

Aspegren, Hunter Goldman, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhota, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabisa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Juber Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vítor Albiero, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaofang Wang, Xiaojuan Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. The llama 3 herd of models, 2024.

- [7] Jiahao Fan, Yi Li, Shaohua Wang, and Tien N. Nguyen. A c/c++ code vulnerability dataset with code changes and cve summaries. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR '20, page 508–512, New York, NY, USA, 2020. Association for Computing Machinery.

- [8] Youdi Gong, Guangzhen Liu, Yunzhi Xue, Rui Li, and Lingzhong Meng. A survey on dataset quality in machine learning. *Information and Software Technology*, 162:107268, 2023.
- [9] Soneya Binta Hossain, Nan Jiang, Qiang Zhou, Xiaopeng Li, Wen-Hao Chiang, Yingjun Lyu, Hoan Nguyen, and Omer Tripp. A deep dive into large language models for automated bug localization and repair. *Proc. ACM Softw. Eng.*, 1(FSE), July 2024.
- [10] Sihao Hu, Tiansheng Huang, Fatih İlhan, Selim Furkan Tekin, and Ling Liu. Large language model-powered smart contract vulnerability detection: New perspectives, 2023.
- [11] Abhinav Jain, Hima Patel, Lokesh Nagalapatti, Nitin Gupta, Sameep Mehta, Shanmukha Guttula, Shashank Mujumdar, Shazia Afzal, Ruhi Sharma Mittal, and Vitobha Munigala. Overview and importance of data quality for machine learning tasks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 3561–3562, New York, NY, USA, 2020. Association for Computing Machinery.
- [12] Nan Jiang, Kevin Liu, Thibaud Lutellier, and Lin Tan. Impact of code language models on automated program repair. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1430–1442, 2023.
- [13] René Just, Darioush Jalali, and Michael D. Ernst. Defects4j: a database of existing faults to enable controlled testing studies for java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, page 437–440, New York, NY, USA, 2014. Association for Computing Machinery.
- [14] Zhenguang Liu, Peng Qian, Xiang Wang, Lei Zhu, Qinming He, and Shouling Ji. Smart contract vulnerability detection: From pure neural network to interpretable graph feature and expert pattern fusion. In *IJCAI*, pages 2751–2759, 2021.
- [15] Wei Ma, Daoyuan Wu, Yuqiang Sun, Tianwen Wang, Shangqing Liu, Jian Zhang, Yue Xue, and Yang Liu. Combining fine-tuning and llm-based agents for intuitive smart contract auditing with justifications, 2024.
- [16] Niels Mündler, Mark Niklas Müller, Jingxuan He, and Martin Vechev. Code agents are state of the art software testers, 2024.
- [17] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024.

- [18] Felix Stahlberg and Shankar Kumar. Synthetic data generation for grammatical error correction with tagged corruption models. In Jill Burstein, Andrea Horbach, Ekaterina Kochmar, Ronja Laarmann-Quante, Claudia Leacock, Nitin Madnani, Ildikó Pilán, Helen Yannakoudakis, and Torsten Zesch, editors, *Proceedings of the 16th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 37–47, Online, April 2021. Association for Computational Linguistics.
- [19] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.
- [20] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.
- [21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.
- [22] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. Automated program repair in the era of large pre-trained language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1482–1494, 2023.