Analyzing gem5 Commit-Code Inconsistency with LLMs

Elise Song, Aaryan Patel, Guruprasad Viswanathan Ramesh, Jack West, Lance Hartung, Ethan Cecchetti, Kassem Fawaz, Joshua San Miguel, Matthew D. Sinclair
University of Wisconsin-Madison
Madison, WI, US
{elise.song, apatel234, viswanathanr, jwwest, lhartung, cecchetti, kfawaz, jsanmiguel, msinclair}@wisc.edu

I. Introduction

Open-Source Software (OSS) is susceptible to supply chain attacks and vulnerabilities introduced by malicious actors. Recent examples include a backdoor enabling remote-code execution discovered in the XZ Utils project [6] and the deliberate addition of Use-After-Free bugs into the Linux kernel [11]. These vulnerabilities often consist of subtle, well-crafted changes designed to evade a small group of maintainers, and can even be injected by malicious maintainers who have spent years building trust [6].

The gem5 simulator [1], [8] faces similar risks, with potentially far-reaching consequences: vulnerabilities introduced into gem5 may propagate into future hardware designs built on its models, especially given its widespread use across academia, industry, and national labs for early-stage design exploration. Although gem5 incorporates a robust testing infrastructure, the codebase remains difficult to maintain due to its breadth of domains and steady influx of hundreds of commits each year. Subtle vulnerabilities like Spectre [5] or Meltdown [7] may be introduced inadvertently or with malicious intent.

Scaling the project places pressure on the review process managed by around 20 volunteer maintainers, who are not trained to detect security threats. Thus, there is a need for tools that vet commits and flag potential vulnerabilities before integration into the main branch. We develop a *Commit-Code Inconsistency Detection System* using Large Language Models (LLMs), which have demonstrated strong capabilities in code reasoning [9], [3], [4]. Our system identifies inconsistent commit message-code pairs and acts as a first step in detecting vulnerable code in OSSs.

II. Design and Results

We define a commit as inconsistent if it is **vague**, **contradictory**, or **incomplete**. A **vague** commit describes the change but omits key details such as motivation, implementation approach, or impact on APIs and usage. A **contradictory** commit occurs when the message conflicts with the actual code changes. A commit is **incomplete** when it introduces *semantically meaningful*, *non-entailed* changes that are not described in the message. *Semantically meaningful* changes include modifications to functionality or APIs, while *non-entailed* changes introduce new guarantees or risks not implied by the commit message. Vague commits are likely more innocent and can be corrected in the continuous integration (CI) flow, whereas contradictory or incomplete commits may indicate an attacker attempting to disguise a vulnerability as a patch.

LLM	Precision	Recall	F1
Gemini-3.1-Pro	0.618	0.971	0.756
Codestral 2	0.407	0.379	0.393

Table I: Performance of our LLM-based System

Our system uses an LLM that takes the commit message, code modifications, and a system prompt as inputs. We filter out vague commits first, then categorize for contradiction and incompleteness. We manually labeled 200 recent gem5 commits into the three inconsistency categories to create a dataset, and evaluated two instantiations of our system using Gemini-3.1-Pro [2] and Codestral2 [10]. Gemini-3.1-Pro outperforms Codestral2, achieving a 76% F1 score with 62% precision and 97% recall in detecting overall inconsistent commits (Table I). High recall is vital to ensure that few inconsistent commits go undetected. Next, we ran our system with Gemini-3.1-Pro on 11,578 commits spanning 2010–2025. We observe that 24% of total commits are inconsistent, of which 15% are vague, 3% are contradictory, and 9% are incomplete (Figure 1).

III. Conclusion

Our preliminary results show that gem5 has a number of commits with potential issues that require further investigation. We hope to use this talk to gather feedback on how to refine our system to better serve the project's needs. Our goal is to integrate this system into the gem5 CI pipeline to help maintainers identify potential security threats and reduce mainte-

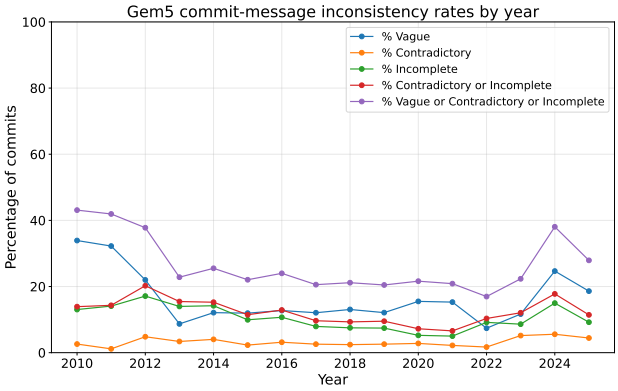

Fig. 1: Commit Inconsistency from 2014-2025 using Gemini-3.1-Pro

nance burden as the project continues to grow.

ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation grant TIP-2533192.

REFERENCES

[1] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[2] Google. Gemini 3.1 pro. https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-1-pro/, 2026. Accessed: 2026-04-23.

[3] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yifan Wu, YK Li, et al. Deepseek-coder: when the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

[4] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swebench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

[5] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[6] Mario Lins, René Mayrhofer, Michael Roland, Daniel Hofer, and Martin Schwaighofer. On the critical path to implant backdoors and the effectiveness of potential mitigation techniques: Early learnings from xz, 2024.

[7] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[8] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Brad Beckmann, Srikant Bharadwaj, Gabe Black, Gedare Bloom, Bobby R. Bruce, Daniel Rodrigues Carvalho, Jeronimo Castrillon, Lizhong Chen, Nicolas Derumigny, Stephan Diestelhorst, Wendy Elsasser, Carlos Escuin, Marjan Fariborz, Amin Farmahini-Farahani, Pouya Fotouhi, Ryan Gambord, Jayneel Gandhi, Dibakar Gope, Thomas Grass, Anthony Gutierrez, Bagus Hanindhito, Andreas Hansson, Swapnil Haria, Austin Harris, Timothy Hayes, Adrian Herrera, Matthew Horsnell, Syed Ali Raza Jafri, Radhika Jagtap, Hanhwi Jang, Reiley Jeyapaul, Timothy M. Jones, Matthias Jung, Subash Kannoth, Hamidreza Khaleghzadeh, Yuetsu Kodama, Tushar Krishna, Tommaso Marinelli, Christian Menard, Andrea Mondelli, Miquel Moreto, Tiago Mück, Omar Naji, Krishnendra Nathella, Hoa Nguyen, Nikos Nikoleris, Lena E. Olson, Marc Orr, Binh Pham, Pablo Prieto, Trivikram Reddy, Alec Roelke, Mahyar Samani, Andreas Sandberg, Javier Setoain, Boris Shingarov, Matthew D. Sinclair, Tuan Ta, Rahul Thakur, Giacomo Travaglini, Michael Upton, Nilay Vaish, Ilias Vougioukas, William Wang, Zhengrong Wang, Norbert Wehn, Christian Weis, David A. Wood, Hongil Yoon, and Éder F. Zulian. The gem5 simulator: Version 20.0+, 2020.

[9] Baptiste Roziere, 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. *arXiv preprint arXiv:2308.12950*, 2023.

[10] Mistral AI Team. Codestral. https://mistral.ai/news/codestral, May 2024. Accessed: 2026-04-23.

[11] Qiushi Wu and Kangjie Lu. On the feasibility of stealthily introducing vulnerabilities in open-source software via hypocrite commits. Linux Reviews, 2021.