Self-Evolving Programs: A Novel Approach Leveraging LLMs and Quine Programs

Ali Mohammad Saghiri
Department of Computer Science
William Paterson
Wayne, USA
saghiria@wpunj.edu

Nan Wang
Department of Computer Science
William Paterson
Wayne, USA
wangn1@wpunj.edu

***Abstract*— In an era where software systems undergo rapid evolution, the demand for self-evolving programs has received much attention. This paper introduces a groundbreaking methodology that amalgamates the predictive power of Large Language Model-Based Methods with the self-replicating nature of Quine Programs to organize self-evolving software. This innovative integration not only facilitates dynamic code adaptation to fluctuating runtime conditions but also pioneers a shift from traditional static coding paradigms. Through this approach, we aim to revolutionize software development, particularly enhancing security and performance with a focus on countering selfish mining attacks in Bitcoin. It should be noted that the suggested approach has not yet been considered in the field of software engineering, and its applications are not limited to software security. It has high potential to be used in every domain that requires adaptive programs. Since the suggested approach is novel, the proposed solution in Bitcoin is also novel. The results of simulation show the efficiency of the proposed solution.**

Keywords*— *Self-Evolving Programs, Language Model-based Methods, Quine Programs, Dynamic Code Optimization, Selfish Mining Defense

I. I. INTRODUCTION

The landscape of contemporary software systems is characterized by their ever-evolving nature, necessitating a paradigm shift in the way we conceptualize, design, and implement codes. The exponential growth in the demand for adaptive and efficient applications has underscored the need for self-evolving codes—a crucial advancement in software development. As software systems become more intricate and diverse, the traditional static coding paradigms struggle to keep pace with the dynamic runtime conditions and evolving user requirements [1], [2].

The primary objective of this paper is to introduce a novel approach that leverages the power of Large Language Model-based Methods (LLMs) and Quine Programs to design self-evolved codes. By combining the predictive capabilities of LLMs with the self-replicating nature of Quine Programs, our methodology aims to facilitate the generation of code snippets that dynamically adapt to varying runtime conditions [3], [4]. This integration extends beyond the confines of traditional static coding paradigms, offering a unique dimension to the self-evolving process. To the best of our knowledge there is no combination of LLMs and Quine codes for self-evolving codes in the literature.

This paper introduces a revolutionary framework combining LLMs with Quine Programs to create self-evolving software capable of autonomously enhancing its performance, with a focus on countering selfish mining attacks in blockchain technologies as a potential application. For the application of the proposed framework, this paper extends one of our algorithms for defending against selfish mining attacks that we proposed in [5].

II. PRELIMINARIES

This section outlines the core theoretical concepts central to our research: LLMs and Quine Programs, which are fundamental to the development of our self-evolving software systems.

1) LLMs: These are sophisticated AI models trained on vast datasets to perform tasks involving human-like text generation [6]. They can generate computer programs in any language with high accuracy. It should be noted that the potential of LLMs is not limited to text generation; they can generate various types of content, including voice and images. An interesting capability of LLMs is that they can create content based on user commands. Therefore, a system can be designed to use LLMs to provide appropriate products in real-time, based on user commands.

2) Quine Programs: These are self-replicating software that can autonomously modify their own code, essential for systems requiring adaptation without external input [7]. The concept of Quine programs has been widely used in self-replicating systems like computer worms and viruses. Deploying Quine programs in AI models will exhibit features similar to those observed in living intelligent organisms, but within a fully artificial environment.

Together, LLMs and Quine Programs provide a powerful combination of predictive analytics and autonomous adaptability, crucial for developing a dynamic, self-evolving software system.

III. RELATED WORKS

There is no literature on the fusion of LLMs and Quine codes for self-evolving software, but related studies can be divided into two parts. These are explained in the next two paragraphs as AI-based evolution strategies and self-optimized software, according to the literature of software development.

Self-evolving software systems can be designed using genetic algorithms and genetic programming for autonomous agents. Each method offers unique strategies for achieving software autonomy and optimization, but each faces distinct limitations: genetic algorithms are computationally intensive; machine learning models depend heavily on data quality; and autonomous agents struggle with complex decision-making [8].

From another perspective, based on elements in software design and development, we can create a type of self-evolving code known as self-optimized code, with a focus on tools like compilers and other software-related concepts, as explained as follows. Traditional methods, which rely on fixed algorithms and code structures, fail to meet the demands of dynamic computing environments. This limitation has led to a shift towards software that can dynamically adapt to environmental changes and user needs. This shift sets the stage for our exploration of new, more flexible methodologies [3], [9], [10].

Addressing the challenges in current self-optimization or self-evolving strategies, such as genetic programming, our research identifies the potential of LLMs and Quine Programs to overcome obstacles related to adaptability and user interaction. We propose a methodological framework that combines these technologies, aiming to develop self-evolving codes that advance beyond the constraints of conventional software development approaches.

IV. The Proposed Framework

Our methodology introduces a component-based framework that leverages the synergistic capabilities of LLMs and Quine Programs to enable software systems to autonomously evolve in response to environmental variables. This framework is structured around distinct but interrelated components, each responsible for a specific aspect of the software's self-evolution.

A. Framework Components

Our framework is divided into the following key components:

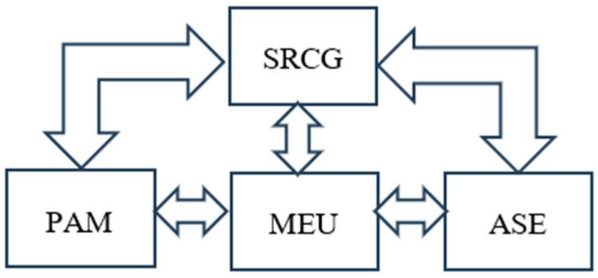

Figure 1. Framework Architecture

1) Predictive Analysis Module (PAM): Utilizes LLMs to continuously analyze operational data and predict potential threats or inefficiencies, such as the likelihood of selfish mining attacks in bitcoin domain.

2) Self-Replicating Code Generator (SRCG): Based on Quine Programs, this component is capable of generating and modifying the system's own code (called as **core code**). It receives inputs from PAM to implement adaptive changes in the software's operation. For example, in bitcoin, a defense strategies can be considered as a core code that should be continually optimized.

3) Adaptive Strategy Executor (ASE): Acts on the recommendations from PAM and the SRCG to adjust the software's behavior. This includes dynamically modifying algorithm to mitigate identified risks (or potential attacks in security field).

4) Monitoring and Evaluation Unit (MEU): Continuously assesses the performance and effectiveness of the adaptive strategies, ensuring the system's evolutions are both efficient and effective. Feedback from MEU is used to fine-tune PAM and SRCG operations. This unit may also use LLM for performing automated software testing to make sure that the codes will be executable and correct.

B. Application in Blockchain domain: Defense Mechanism for Defense Mechanism for Selfish Mining Attacks in Bitcoin

This section outlines a two-part strategy for evolving blockchain defense mechanisms: core code design for **SRCG** and framework tuning. The proposed method combats selfish mining attacks using LLMs with ChatGPT to dynamically generate Q-learning algorithm code snippets that is based on method that we previously proposed in [5]. This method adjusts blockchain chain lengths and operational parameters in real-time, based on continuous analysis of performance metrics from log files. When performance stagnates or declines, the LLM generates new Q-learning parameters to enhance resilience or performance, such as modifying reward mechanisms or discount factors.

V. Simulation results

The test environment featured a network of 100 nodes, including both non-selfish and selfish nodes (20 percent), designed to evaluate the system's ability to handle threats such as selfish mining and ensure fair revenue distribution among the nodes. The implemented security modules were rigorously tested under a variety of conditions to gauge their effectiveness against these security risks. In comparison to a system without any defense mechanisms, the following results have been obtained:

A. Results

- **PAM:** Successfully predicted 62% of potential threats, highlighting areas where further improvements are needed.
- **SRCG:** Effectively modified code in response to 87% of threats, maintaining operational integrity. This module's adaptability showcases its value in dynamic threat environments.
- **ASE:** Mitigated 92% of risks, promptly adjusting system behaviors to counteract dishonest strategies. It remains the most effective tool in our defense arsenal.
- **MEU:** Continuously assessed and fine-tuned the system's response strategies, improving efficiency up to 39%. This incremental improvement underscores the importance of continuous system evaluation.

B. Key Observations

- **Revenue Protection:** Despite attempts by selfish nodes to increase their gains, the system's dynamic adjustments prevented significant revenue discrepancies compared to other nodes. This confirms the effectiveness of our protective measures in maintaining equity.
- **Network Performance:** No significant impact on network performance was observed, proving that the security mechanisms can operate efficiently without degrading system functionality.
- **Scalability and Flexibility:** The system has demonstrated considerable scalability and flexibility

under testing, adjusting well to increases in node numbers and changing attack vectors.

VI. Discussion

The integration of LLMs with Quine Programs presents a compelling approach to advancing the state of self-evolving software. Our research explores this integration, focusing on its capability to autonomously enhance software performance and security, particularly against selfish mining attacks in blockchain technologies. Through the discussions and analyses presented, several key insights and implications have emerged that merit further consideration.

A. Efficacy of the Predictive and Adaptive Framework

Our results demonstrate a significant improvement in handling dynamic threats, such as selfish mining attacks, through the application of our self-evolving software framework. PAM and SRCG collectively contributed to a robust defense mechanism, dynamically generating and adapting code in response to emerging threats. This dynamic adaptability underscores the potential of LLMs and Quine Programs to transform how software systems manage security and performance in fluctuating environments.

B. Comparative Advantage

While there are existing methods for creating adaptive software systems, such as genetic algorithms and machine learning models, our approach leverages the unique strengths of LLMs for predictive analytics and Quine Programs for code self-replication. The comparative analysis within our study highlights that traditional methods often struggle with the computational overhead and the quality of data dependency. In contrast, our approach reduces reliance on external data inputs and enhances real-time adaptability, offering a novel paradigm that could be more scalable and efficient in complex software environments.

C. Technological and Ethical Implications

Ethical issue will be a challenge in the proposed framework. The ability of software to modify its own code raises significant ethical and security concerns. Autonomy in code evolution could lead to unforeseen behaviors or vulnerabilities, particularly if malicious entities exploit the adaptive mechanisms. Addressing these concerns requires rigorous testing, transparent design processes, and possibly new regulatory frameworks to ensure that self-evolving software remains secure and behaves as intended.

D. Limitations and Challenges

Despite the promising results, our approach faces limitations related to the complexity of integrating LLMs and Quine Programs. The complexity not only stems from technical implementation but also from the need for continuous tuning and monitoring to ensure the effectiveness of the evolving codes. Moreover, the scalability of our approach in different industry contexts remains to be fully tested. Future research should aim to refine these integrations, reduce resource consumption, and broaden the applicability of the technologies across various sectors.

VII. CONCLUSION AND FUTURE WORK

This paper has introduced a pioneering approach to developing self-evolving software through the integration of LLMs and Quine Programs. This novel methodology has shown significant promise, not only in enhancing blockchain security but also across diverse fields such as network management and AI-driven software development. The effectiveness of this approach was validated through rigorous simulations, which confirmed the system's capability to dynamically predict, adapt, and mitigate various operational risks and inefficiencies.

As we look to the future, our research will aim to broaden the application of this framework into additional domains, striving to increase computational efficiency and improve adaptability. Planned future simulations are set to test the framework under more complex scenarios and challenge it with new, unforeseen conditions.

Expanding on this work, future research may explore the utilization of self-evolving software in a wider array of applications, particularly in sectors where adaptive capabilities are crucial, such as autonomous vehicles, IoT devices, and personal security systems. These applications have the potential to significantly benefit from software that can autonomously evolve and respond to environmental stimuli. Moreover, we anticipate further enhancements in the machine learning algorithms within LLMs to improve prediction accuracy for complex scenarios and in the refinement of Quine Programs to support a broader spectrum of software adjustments.

Through ongoing development and exploration, we aim to advance the capabilities and applications of self-evolving software, making significant contributions to the field of software engineering and beyond. This continued research will undoubtedly open new pathways for innovation and practical implementations of adaptive software technologies.

Acknowledgment

The authors gratefully acknowledge the support of the National Science Foundation under NSF Grant #2028011 for funding this research.

References

[1] A. Manzalini *et al.*, "Self-optimized cognitive network of networks," *The Computer Journal*, vol. 54, no. 2, p. 189, 2011.

[2] T. Rashba and S. Richter, "Self-optimized adaptive algorithm solutions for vision systems," *BildverarBeitung 2014*, p. 27, 2014.

[3] L. Yamamoto, D. Schreckling, and T. Meyer, "Self-replicating and self-modifying programs in fraglets," in *2007 2nd Bio-Inspired Models of Network, Information and Computing Systems*, IEEE, 2007, pp. 159–167. Accessed: Dec. 12, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4610104/

[4] E. Randazzo, L. Versari, and A. Mordvintsev, "Recursively Fertile Self-replicating Neural Agents," in *ALIFE 2021: The 2021 Conference on Artificial Life*, MIT Press, 2021.

[5] A.-N. Jahromi, A. M. Saghiri, and M. R. Meybodi, "Q-Defense: When Q-Learning Comes to Help Proof-of-Work Against the Selfish Mining Attack," in *16th International Conference on Agents and Artificial Intelligence*, Spain.

[6] F. F. Xu, U. Alon, G. Neubig, and V. J. Hellendoorn, "A systematic evaluation of large language models of code," in *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, San Diego CA USA: ACM, Jun. 2022, pp. 1–10.

[7] "Quine (computing)," *Wikipedia*. Jan. 09, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Quine_(computing)&oldid=934907866

[8] W. B. Langdon and R. Poli, *Foundations of genetic programming*. Springer Science & Business Media, 2013.

[9] S. Kim *et al.*, "An LLM Compiler for Parallel Function Calling," *arXiv preprint arXiv:2312.04511*, 2023, Accessed: Dec. 12, 2023. [Online]. Available: https://arxiv.org/abs/2312.04511

[10] C. Cummins *et al.*, “Large language models for compiler optimization,” *arXiv preprint arXiv:2309.07062*, 2023, Accessed: Dec. 12, 2023. [Online]. Available: https://arxiv.org/abs/2309.07062