American Journal Of Applied Science And Technology
(ISSN – 2771-2745)
VOLUME 04 ISSUE 11 Pages: 106-113
OCLC – 1121105677

Publisher: Oscar Publishing Services

Journal Website: https://theusajournals.com/index.php/ajast

Research Article

The Convergence of Machine Learning and Large Language Models in Software Architecture: A Comprehensive Analysis of Defect Prediction, Design Patterns, And Automated Recovery

Submission Date: October 26, 2024, Accepted Date: November 17, 2024,
Published Date: November 30, 2024

Jiya Nelson
Department of Software Engineering, University of Melbourne, Australia

ABSTRACT

The integration of Machine Learning (ML) and Large Language Models (LLMs) into the software development lifecycle has catalyzed a paradigm shift in how architectural integrity and code quality are maintained. This research article provides an extensive investigation into the evolution of automated software engineering, spanning from classical ML-based defect prediction and code smell detection to contemporary LLM-driven architectural recovery and design pattern adoption. By synthesizing evidence from systematic mapping studies and recent empirical evaluations, the study explores the transition from discriminative models-used for identifying package-level clones and software vulnerabilities-to generative architectures capable of end-to-end program repair and high-level component summarization. The research further examines the efficacy of Chain-of-Thought (CoT) prompting and abstract syntax tree (AST) representations in enhancing the semantic understanding of complex codebases. Through a detailed analysis of cross-project defect prediction, modularization of legacy systems, and the extraction of architectural information from informal specifications, this article highlights the transformative potential of AI-driven practices. The findings suggest that while LLMs possess significant architectural knowledge, their integration requires precise probing techniques and formal rule learning to mitigate risks associated with hallucinations and maintain conformance in continuous integration environments.

KEYWORDS

Machine Learning, Large Language Models, Software Architecture, Defect Prediction, Code Smells, Software Recovery, Generative AI.

INTRODUCTION

American Journal Of Applied Science And Technology
(ISSN – 2771-2745)
VOLUME 04 ISSUE 11 Pages: 106-113
OCLC – 1121105677
Crossref doi Google Scholar WorldCat MENDELEY

Publisher: Oscar Publishing Services

The modern software engineering landscape is characterized by an exponential increase in complexity, necessitating robust mechanisms for maintaining code quality and architectural coherence. Traditionally, software maintenance and evolution relied heavily on manual inspection and heuristic-based tools. However, the emergence of Machine Learning (ML) as a foundational pillar in software analytics has introduced a more data-driven approach to identifying structural flaws, known as code smells, and predicting latent defects before they manifest in production environments. As software systems grow into massive, interconnected ecosystems, the ability to automatically detect package-level clones and violations of design patterns becomes critical for long-term sustainability.

In recent years, the field has moved beyond simple classification tasks toward a more nuanced understanding of code as a language. This transition has been spearheaded by the advent of Large Language Models (LLMs) and deep learning architectures that leverage multi-modal representations of source code. Unlike traditional ML models that may rely on static metrics such as cyclomatic complexity or lines of code, modern deep learning approaches utilize Abstract Syntax Trees (ASTs) and tree-based neural models to capture the hierarchical and semantic structures of programs. This allows for more sophisticated operations, such as automated code summarization, natural language elicitation of requirements, and the re-engineering of legacy systems.

The problem, however, remains multifaceted. While ML techniques have shown promise in specific domains like defect identification and design pattern detection, they often struggle with the "cold start" problem in new projects where historical data is scarce. This has led to the development of cross-project defect prediction (CPDP) and deep transfer learning techniques, which attempt to leverage knowledge from mature repositories to improve the reliability of newer software assets. Simultaneously, the rise of Generative AI (GenAI) introduces new challenges and opportunities in software architecture. Architectural recovery-the process of extracting high-level design intent from low-level implementation-has been historically difficult. Modern research now explores whether LLMs can "reason" through these architectures using deductive methods and hierarchical prompting strategies.

This article aims to bridge the gap between classical ML applications in software quality and the cutting-edge deployment of LLMs in architectural design and conformance. By analyzing the current state of the art, we explore how these technologies can be harmonized to create a self-healing, self-documenting software development lifecycle. The literature reveals a clear trajectory from simple "bug finding" to complex "architectural reasoning," yet significant gaps remain regarding the formalization of these AI-driven rules and the precision of the information they extract.

METHODOLOGY

The methodology of this research is grounded in a comprehensive synthesis of current literature, utilizing a multi-layered analytical framework to evaluate the efficacy of ML and LLM techniques across the software engineering spectrum. The study adopts a systematic mapping approach to categorize various AI-driven interventions based on their functional objectives: detection, prediction, summarization, and recovery.

To understand the evolution of defect prediction, the research examines the performance of various ML algorithms, including support vector machines,

American Journal Of Applied Science And Technology
(ISSN – 2771-2745)
VOLUME 04 ISSUE 11 Pages: 106-113
OCLC – 1121105677
Crossref doi Google Scholar WorldCat MENDELEY

Publisher: Oscar Publishing Services

random forests, and deep neural networks, across diverse datasets. The analysis focuses on how feature engineering-ranging from traditional software metrics to latent representations learned through deep learning-impacts the accuracy of defect identification. Particular attention is given to the methodology of cross-project defect prediction, where the challenge of feature distribution mismatch is addressed through alignment techniques and transfer learning.

In the realm of code smells and design patterns, the methodology involves investigating how ML models are trained to recognize patterns of "poor" design that lead to technical debt. This includes a review of systematic mapping studies that identify the most prevalent ML techniques used for code smell detection, such as clustering and classification. The research evaluates the transition from manual rule-based detection to automated learning, where models are exposed to thousands of labeled examples of design violations to develop a "probabilistic intuition" for code quality.

The evaluation of LLMs in software architecture necessitates a different methodological lens. Here, the research focuses on "probing" and "prompting" strategies. We analyze the use of Chain-of-Thought (CoT) prompting, where the model is guided through a step-by-step reasoning process to recover architectural components or summarize complex modules. The methodology also includes a comparative investigation between LLMs and inductive techniques for learning formal architecture rules. By examining case studies involving legacy system re-engineering (such as those conducted within the Volvo Group), the research assesses the practical utility of GenAI in real-world, large-scale industrial contexts.

Finally, the study investigates multi-modal learning representations. This involves understanding how combining different views of code-such as the raw text, the AST, and the control flow graph-can lead to more robust models for code editing and repair. The methodology synthesizes findings from experiments using sequence-to-sequence learning and tree-based neural models to determine the optimal balance between textual and structural information in program synthesis and repair tasks.

RESULTS

The descriptive analysis of current research findings reveals several critical insights into the performance and adoption of AI in software engineering. In the domain of software defect prediction, results indicate that ML-based systems consistently outperform traditional static analysis in identifying high-risk modules. Empirical assessments show that while no single algorithm is a "silver bullet," ensemble methods and deep learning architectures like DeepCPDP show significant improvements in F1-scores and precision, especially in the context of cross-project scenarios where target data is limited.

Regarding code smell and design pattern detection, the literature demonstrates that ML techniques have moved beyond the experimental phase and are increasingly capable of identifying complex violations that span multiple classes or packages. Mapping studies confirm that the use of unsupervised learning is particularly effective for discovering new types of code smells that were not previously codified by human experts. Furthermore, the application of deep transfer learning has proven effective in software visualization, allowing developers to see a "heat map" of potential defects based on the latent features learned by the model.

American Journal Of Applied Science And Technology
(ISSN – 2771-2745)
VOLUME 04 ISSUE 11 Pages: 106-113
OCLC – 1121105677
Crossref doi Google Scholar WorldCat MENDELEY
Publisher: Oscar Publishing Services

In the sphere of LLMs and Generative AI, the results suggest a high degree of architectural knowledge embedded within models like GPT-4 and its successors. Research into hierarchical CoT prompting shows that LLMs can achieve high-level software component summarization with a degree of accuracy that rivals human architects. In tasks involving deductive software architecture recovery, LLMs have demonstrated the ability to infer the underlying structure of a system purely from source code and informal specifications. However, the results also highlight a "precision gap." While LLMs are excellent at generating plausible-sounding architectural descriptions, they require precise "probing questions" to elicit deep architectural information and avoid superficial summaries.

The re-engineering of legacy systems remains one of the most promising yet challenging results of recent AI adoption. Experiences from industrial case studies indicate that LLMs can significantly accelerate the modularization of monolithic systems by suggesting service boundaries and identifying design pattern candidates. However, the results also show that formal architecture rule learning is still a domain where inductive techniques sometimes provide more reliable, verifiable constraints than the probabilistic outputs of LLMs.

Finally, the results of code editing and program repair studies indicate that tree-based neural models, such as CODIT, offer superior performance in generating syntactically correct code edits compared to simple sequence-to-sequence models. By leveraging the AST, these models ensure that the proposed repairs honor the structural rules of the programming language, leading to a higher rate of successful compilations and functional fixes.

DISCUSSION

The transition from classical ML to generative LLMs in software engineering represents more than just a technological upgrade; it is a fundamental change in the relationship between the developer and the codebase. The discussion must first address the implications of ML-based defect prediction. While the accuracy of these models is high, the "black box" nature of deep learning remains a hurdle for developer trust. If a model predicts a defect in a critical module, the developer needs to understand why. This is where the integration of software visualization and explainable AI becomes paramount. The findings suggest that the most successful implementations are those that pair prediction with clear visualizations of the underlying metrics causing the alert.

A significant point of discussion is the modularization of legacy systems. Many organizations struggle with "spaghetti code" that has evolved over decades. The use of ML-assisted service boundary detection offers a systematic way to break down these monoliths. However, as noted in recent studies, the detection of boundaries is as much a social and business task as it is a technical one. The AI can suggest where to cut, but it cannot always account for the organizational structures or domain-driven design nuances that a human architect considers. Therefore, the role of the AI is best viewed as a "recommender" rather than a "decider."

The emergence of LLMs in architectural design brings about the concept of "informal specifications." Historically, formalizing requirements has been a bottleneck in the software lifecycle. LLMs allow for the automation of extracting key information from natural language requirements using techniques like the "problem frames" approach. This democratization of architectural design means that stakeholders with less technical backgrounds can participate more effectively

American Journal Of Applied Science And Technology
(ISSN – 2771-2745)
VOLUME 04 ISSUE 11 Pages: 106-113
OCLC – 1121105677
Crossref doi Google Scholar WorldCat MENDELEY
Publisher: Oscar Publishing Services

in the design process. Yet, this introduces the risk of architectural drift, where the implementation slowly diverges from the AI-generated design. Continuous architectural conformance tools, powered by LLMs, are being developed to resolve these violations in real-time within the CI/CD pipeline.

Furthermore, we must discuss the limitations of GenAI in software development. The reliance on CoT prompting suggests that LLMs are not inherently "aware" of software architecture in the way a human is; rather, they are highly sophisticated pattern matchers. The "knowledge" they possess is a reflection of the millions of lines of code and architectural documents they were trained on. This leads to questions of originality and the potential for reinforcing outdated or sub-optimal design patterns if those patterns are prevalent in the training data. The comparative investigation between LLMs and inductive techniques highlights that for high-stakes, formal rule enforcement, we may still need the rigor of traditional logic-based methods.

The future scope of this research lies in the "multi-modal" future. As models become better at handling multiple input representations-raw code, ASTs, documentation, and even visual diagrams of system architecture-we will see a more holistic AI assistant. This assistant will not just find bugs or summarize classes; it will understand the "intent" of the software. It will be able to explain how a change in a low-level API might ripple through the system and violate a high-level architectural constraint established months prior.

CONCLUSION

This research has traversed the expansive territory of Machine Learning and Large Language Models as they apply to the critical domains of software architecture and quality. We have seen that the journey begins with the precision of ML in detecting code smells and predicting defects, providing a foundational layer of automated quality assurance. The evolution into deep learning and AST-based representations has further refined our ability to summarize code and repair programs with high structural fidelity.

The integration of LLMs marks the latest frontier, offering unprecedented capabilities in architectural recovery, service boundary detection, and natural language requirement extraction. These models, when guided by sophisticated prompting strategies like Chain-of-Thought, can act as powerful partners in re-engineering legacy systems and maintaining architectural conformance. However, the study concludes that AI is not a replacement for the human architect. Instead, it is a force multiplier that requires precise elicitation, formal verification, and a deep understanding of the underlying software engineering principles.

Ultimately, the synergy between data-driven ML models and knowledge-rich LLMs provides a path forward for managing the increasing complexity of modern software. By leveraging these tools, organizations can reduce technical debt, improve system reliability, and accelerate the transformation of legacy codebases into modern, modular architectures. The challenge for the next generation of software engineers will be to master the art of "probing" these models to extract the maximum architectural value while maintaining the rigorous standards of formal software design.

REFERENCES

1. Caram Frederico Luiz, Rodrigues Bruno Rafael De Oliveira, Campanelli Amadeu Silveira, Parreiras Fernando Silva. Machine learning techniques for code smells detection: A systematic mapping

American Journal Of Applied Science And Technology
(ISSN – 2771-2745)
VOLUME 04 ISSUE 11 Pages: 106-113
OCLC – 1121105677
Crossref doi Google Scholar WorldCat MENDELEY
Publisher: Oscar Publishing Services

study. Int. J. Softw. Eng. Knowl. Eng., 29 (02) (2019), pp. 285-316, 10.1142/S021819401950013X.

2. Cesare Silvio, Xiang Yang, Zhang Jun. Clonewise – detecting package-level clones using machine learning. Zia Tanveer, Zomaya Albert, Varadharajan Vijay, Mao Morley (Eds.), Security and Privacy in Communication Networks, 978-3-319-04283-1 (2013), pp. 197-215.
3. Cetiner M., Sahingoz O.K. A comparative analysis for machine learning based software defect prediction systems. 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT) (2020), pp. 1-7, 10.1109/ICCCNT49239.2020.9225352.
4. Ceylan E., Kutlubay F.O., Bener A.B. Software defect identification using machine learning techniques. 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06) (2006), pp. 240-247, 10.1109/EUROMICRO.2006.56.
5. Chakraborty S., Ding Y., Allamanis M., Ray B. CODIT: Code editing with tree-based neural models. IEEE Trans. Softw. Eng. (2020), p. 1, 10.1109/TSE.2020.3020502.
6. Chakraborty Saikat, Ding Yangruibo, Allamanis Miltiadis, Ray Baishakhi. CODIT: Code editing with tree-based neural models. IEEE Trans. Softw. Eng., 48 (4) (2022), pp. 1385-1399, 10.1109/TSE.2020.3020502.
7. Chakraborty Saikat, Ray Baishakhi. On multi-modal learning of editing source code. 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE) (2021), pp. 443-455, 10.1109/ASE51524.2021.1003_Chakraborty2021.
8. Challagulla Venkata Udaya B., Bastani Farokh B., Yen I-Ling, Paul Raymond A. Empirical assessment of machine learning based software defect prediction techniques. Int. J. Artif. Intell. Tools, 17 (02) (2008), pp. 389-400, 10.1142/S0218213008003947.
9. Chappelly T., Cifuentes C., Krishnan P., Gevay S. Machine learning for finding bugs: An initial report. 2017 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE) (2017), pp. 21-26, 10.1109/MALTESQUE.2017.7882012.
10. Chaturvedi Shivam, Chaturvedi Amrita, Tiwari Anurag, Agarwal Shalini. Design pattern detection using machine learning techniques. 2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO), IEEE (2018), pp. 1-6.
11. Chen Deyu, Chen Xiang, Li Hao, Xie Junfeng, Mu Yanzhou. Deepcpdp: Deep learning based cross-project defect prediction. IEEE Access, 7 (2019), pp. 184832-184848.
12. Chen Qiuyuan, Hu Han, Liu Zhaoyi. Code summarization with abstract syntax tree. Gedeon Tom, Wong Kok Wai, Lee Minho (Eds.), Neural Information Processing, 978-3-030-36802-9 (2019), pp. 652-660.
13. Chen Jinyin, Hu Keke, Yu Yue, Chen Zhuangzhi, Xuan Qi, Liu Yi, Filkov Vladimir. Software visualization and deep transfer learning for effective software defect prediction. Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, ICSE '20, 9781450371216 (2020), pp. 578-589, 10.1145/3377811.3380389.
14. Chen Fuxiang, Kim Mijung, Choo Jaegul. Novel natural language summarization of program code via leveraging multiple input representations. Findings of the Association for Computational Linguistics: EMNLP 2021, Association for Computational Linguistics, Punta Cana, Dominican Republic (2021), pp. 2510-2520, 10.18653/v1/2021.findings-emnlp.214.

American Journal Of Applied Science And Technology
(ISSN – 2771-2745)
VOLUME 04 ISSUE 11 Pages: 106-113
OCLC – 1121105677
Crossref doi Google Scholar WorldCat MENDELEY
Publisher: Oscar Publishing Services

15. Chen Z., Kommrusch S.J., Tufano M., Pouchet L., Poshyvanyk D., Monperrus M. SEQUENCER: Sequence-to-sequence learning for end-to-end program repair. IEEE Trans. Softw. Eng. (2019), p. 1, 10.1109/TSE.2019.2940179.
16. Chen Xinyun, Liu Chang, Shin Richard, Song Dawn, Chen Mingcheng. Latent attention for if-then program synthesis. Proceedings of the 30th International Conference on Neural Information Processing Systems, NIPS '16, 9781510838819 (2016), pp. 4581-4589.
17. Prompts_PlantUML_Similarity_Score. https://github.com/renjith-r02/prompts-plantuml-similarityscore/blob/main/Prompts_PlantUML_Similarity_Score_v2.pdf.
18. P. Raghavan. Ipek ozkaya on generative ai for software architecture. IEEE Softw., 41 (2024), pp. 141-144.
19. Rejithkumar, G., Anish, P.R., Shukla, J., Ghaisas, S., 2024. Probing with precision: Probing question generation for architectural information elicitation. In: 2024 IEEE/ACM Workshop on Multi-Disciplinary, Open, and RElevant Requirements Engineering. MO2RE, pp. 8–14.
20. R. Rubei, A. Di Salle, A. Bucaioni. Llm-based recommender systems for violation resolutions in continuous architectural conformance. 2025 IEEE 22nd International Conference on Software Architecture Companion, ICSA-C, IEEE Computer Society, Los Alamitos, CA, USA (2025), pp. 404-409.
21. Rukmono, S.A., Ochoa, L., Chaudron, M.R., 2023. Achieving high-level software component summarization via hierarchical chain-of-thought prompting and static code analysis. In: 2023 IEEE International Conference on Data and Software Engineering. ICoDSE, pp. 7–12.
22. S.A. Rukmono, L. Ochoa, M. Chaudron. Deductive software architecture recovery via chain-of-thought prompting. Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results, ICSE-NIER'24, Association for Computing Machinery, New York, NY, USA (2024), pp. 92-96.
23. L. Saarinen. Generative ai in software development. Inf. Technol. (2024).
24. C. Schindler, A. Rausch. Formal software architecture rule learning: A comparative investigation between large language models and inductive techniques. Electronics, 13 (2024).
25. Sharma, T., 2024. Llms for code: The potential, prospects, and problems. In: 2024 IEEE 21st International Conference on Software Architecture Companion. ICSA-C, pp. 373–374.
26. V. Singh, C. Korlu, O. Orcun, W.K. Assunçao. Experiences on using large language models to re-engineer a legacy system at volvo group. Methods, 13 (2024), p. 14.
27. M. Soliman, J. Keim. Do large language models contain software architectural knowledge? : An exploratory case study with gpt. 2025 IEEE 22nd International Conference on Software Architecture, ICSA, IEEE Computer Society, Los Alamitos, CA, USA (2025), pp. 13-24.
28. V. Supekar, P. MIT WPU, R. Khande. Improving software engineering practices: Ai-driven adoption of design patterns (2024).
29. Tagliaferro, S. Corboe, B. Guindani. Leveraging llms to automate software architecture design from informal specifications. 2025 IEEE 22nd International Conference on Software Architecture Companion, ICSA-C, IEEE Computer Society, Los Alamitos, CA, USA (2025), pp. 291-299.
30. Tang, S., Chen, X., Xiao, H., Wei, J., Li, Z., 2023. Using problem frames approach for key information extraction from natural language requirements. In: 2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security Companion. QRS-C, pp. 330–339.

American Journal Of Applied Science And Technology
(ISSN – 2771-2745)
VOLUME 04 ISSUE 11 Pages: 106-113
OCLC – 1121105677
Crossref doi Google Scholar WorldCat MENDELEY
Publisher: Oscar Publishing Services

31. K. S. Hebbar, "MACHINE LEARNING-ASSISTED SERVICE BOUNDARY DETECTION FOR MODULARIZING LEGACY SYSTEMS," International Journal of Applied Engineering & Technology, vol. 04, no.02, pp. 401-414, Sep. 2022, https://romanpub.com/resources/ijaet-v4-2-2022-48.pdf