Towards Just-In-Time, Inclusive Clone Refactoring

Palash Roy
Dept. of Computer Science
University of Saskatchewan
Saskatoon, Canada
palash.roy@usask.ca

2025 IEEE International Conference on Software Maintenance and Evolution (ICSME) | 979-8-3315-9587-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICSME64153.2025.00099

***Abstract*—Code clones are a well-known source of technical debt, often degrading software maintainability. Modern AI coding assistants can unintentionally introduce clones or even license-incompatible code, posing maintenance and legal challenges. This research proposes a Large Language Model (LLM)-powered clone refactoring assistant that proactively detects duplicative code during development and suggests high-level refactorings. By bridging traditional clone detection with LLM-driven code generation or transformation, the approach aims to remove redundancies early while ensuring changes remain behavior preserving and legally compliant. The assistant will integrate into development workflows to prevent clone propagation, flag potential intellectual property risks, and incorporate developer feedback for explainable, trustworthy operation. We will evaluate its impact on code quality and developer productivity, and assess how AI enhancements influence long-term maintenance efforts.**

***Index Terms*—Code Clone, refactoring, large language model, IDE, Tool, RAG, Ranking, Trustworthy, automatic maintenance**

I. Problem Statement

Code clones or identical or near-duplicate code fragments are common in modern software systems, often comprising 7–24% and sometimes up to 50% of code [5], [18], [19]. While copy-paste reuse can yield short-term gains, unchecked cloning is a known "bad smell" linked to defect propagation and high maintenance costs [8]. This problem has intensified with the rise of AI assistants, now used by 92% of North American developers [10]. Recent industry data show a 4× increase in duplicated code under AI-assisted workflows [9], as engineers often accept generated code "as-is." These trends demand in-context clone management tools that proactively surface and help resolve redundancy during development.

Clone management remains predominantly reactive, with detection and remediation often delayed until later maintenance phases. However, recent research advocates for a proactive strategy, detecting and addressing clones as they are introduced [3], [18], [30]. For instance, CCEvents [29] monitors version control commits and alerts developers to clone creation in near real-time. JDeodorant [7], [24] can suggest Extract Method refactorings for simple Type-1/2 clones, but its functionality is limited and not integrated into a proactive development workflow. It also does not support more complex clone types, allowing technical debt to accumulate. A recent survey by Bharti et al. [3] identified 33 IDE-integrated tools, highlighting increasing interest in real-time, preventive clone management.

Recent advances in machine learning (ML) and deep learning have enhanced clone detection and maintenance [1], [2], [20], [21]. ML-based frameworks use structural features (length, dispersion, uniqueness) to identify refactorable clones and reduce defect density [12], [15], [16]. Deep models can detect semantically equivalent clones, outperforming token and AST-based techniques [22]. Large Language Models (LLMs) offer a promising next step: when combined with retrieval-augmented generation (RAG), they can generate context-aware refactoring suggestions [13], [23]. Tools like CREC [28] and MARC [15] evaluate structural and historical features of clone groups to predict refactorability, with CREC achieving up to 83% F1-score across multiple projects. These systems demonstrate how context and history can improve clone maintenance, laying a foundation for more intelligent, LLM-driven approaches.

Building on these foundations, LLMs offer a promising frontier for automated clone management. Especially when combined with RAG and contextual grounding, LLMs can produce meaningful, context-aware refactoring suggestions tailored to the developer's codebase [13], [23]. Researchers envision IDE-integrated assistants that not only detect clones in real time but also assess potential risk and propose transformations. These modern systems, leveraging deep learning, RAG, and human-in-the-loop design, show strong potential for proactively eliminating redundancy. Many research underscores this trajectory, emphasizing the growing need for intelligent, LLM-assisted clone resolution embedded directly within development workflows [3], [4], [6], [17]

However, significant challenges remain regarding the reliability [20], [26], interpretability [25], and legal safety [27] of such AI-generated recommendations. LLM-based suggestions can produce false positives or even hallucinated code [14], [20], and Xu et al. [27] demonstrate that even minor edits to a copied code snippet may still violate open-source license compliance rules. Clone detection has historically been used to identify reuse of GPL-licensed code in proprietary software [11], and similar analysis can aid in surfacing such risks early when AI assistants introduce recycled code. In particular, AI-generated code must be checked to avoid violating licensing or patent constraints [11], [27]. Thus, building a trustworthy AI-powered clone refactoring assistant requires not only advanced automation but also a human-centered design and rigorous validation strategy to ensure the tool is reliable, interpretable, and legally compliant. Such an approach is crucial to make sure these tools genuinely assist developers rather than becoming another source of noise or risk.

2576-3148/25/$31.00 ©2025 IEEE
DOI 10.1109/ICSME64153.2025.00099

To address these challenges, we propose a LLM-powered clone refactoring assistant that bridges traditional clone detection with modern AI-based transformation. The system will proactively detect clones during development, assess their refactorability based on structural and contextual metrics (size, dispersion, and uniqueness), and generate actionable refactoring suggestions using RAG). To guide our investigation, we aim to answer the following research questions:

RQ1: How can proactive clone management be aligned with developer practices and integrated into modern IDE workflows?

RQ2: How do AI enhance automated clone detection and refactoring, and what are the key limitations of these approaches?

RQ3: What benefits does proactive clone management provide in terms of software quality and maintainability?

RQ4: How can clone management systems be extended to detect and mitigate license and intellectual property risks in code reuse?

RQ5: What role should developer feedback and explainability play in fostering trust in AI-driven refactoring tools?

II. Expected Outcome

This research will produce a prototype of an LLM-powered assistant that proactively detects clones, analyzes their contextual relevance, and suggests legally compliant refactorings. We expect to show that integrating such a system into the development workflow enables earlier clone resolution, reduces redundant code, and improves long-term maintainability. Through empirical validation and developer-in-the-loop studies, the project will yield practical insights into the effectiveness, trustworthiness, and usability of AI-assisted clone management. Outcomes will include a validated prototype, metrics for clone refactorability, and implementation guidelines for integrating such assistants into modern IDEs.

III. Expected Contribution

Our work contributes a novel, AI-augmented framework for proactive clone management by unifying classical detection with LLM-based, context-aware refactoring and legal compliance checks. By leveraging RAG and embedding safeguards against license violations, the assistant advances the state of the art in trustworthy, semi-automated software maintenance. Contributions include: (1) a fully integrated IDE-based tool, (2) empirical evaluations of usability and technical accuracy, (3) refined models of clone refactorability, and (4) actionable design principles for human-in-the-loop AI systems. Together, these contributions aim to reshape clone management into a proactive, compliant, and developer-aligned practice.

IV. Planned Evaluation

We will evaluate the approach through a combination of quantitative and qualitative studies, aligned with our research questions:

Workflow Integration and Impact (RQ1 & RQ3): We will integrate the proactive clone management tool into modern development workflows (as an IDE plugin) to observe its effectiveness, tracking how early clone detection and refactoring influence code quality and maintenance effort. We will measure outcomes such as reduced duplication, fewer clone related bugs, and lower maintenance effort by comparing projects with and without the tool.

AI Enhancements and License Risks (RQ2 & RQ4): To evaluate the benefits of LLM assistance, we will compare our AI-augmented clone refactoring suggestions against baseline techniques on benchmarks. This will quantify improvements in detection recall and suggestion correctness while revealing any limitations. We will also extend the system to flag clones that may violate license or IP constraints, validating this feature on known license-violating clones and measuring its detection accuracy against license-scanning tools.

Developer Feedback and Trust (RQ5): We will conduct a user study where developers use the refactoring assistant with and without explanation of its suggestions. We will gather qualitative feedback (interviews) and quantitative data (task completion times, error rates, trust questionnaires) to assess how explainability and feedback influence developer confidence and tool adoption. These results will guide further refinements to foster trust in the AI-driven refactoring assistant.

V. Limitations

This early-stage research faces several limitations. First, our initial evaluation will be focused on Python and Java, which may limit programming language generalizability. Future work will expand language coverage. Second, to preserve IDE responsiveness, clone detection will be triggered on file-save rather than continuously, which may delay feedback. We plan to explore lightweight, incremental detection to balance precision and performance. Third, while RAG enhances LLM suggestion quality, hallucinations and irrelevant refactorings remain a risk. We will mitigate this via automated validation (compilability, test execution) and prompt refinement. Lastly, our user study may be limited in size due to resource constraints. To ensure rigor, we will employ within-subject designs, validated instruments (SUS, NASA-TLX), and qualitative interviews. As the work evolves, we remain open to refining the evaluation design and research questions based on empirical findings.

Acknowledgment

This research is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grants program, the Canada Foundation for Innovation's John R. Evans Leaders Fund (CFI-JELF), and the industry-stream NSERC CREATE in Software Analytics Research (SOAR). I would also like to thank my supervisor, Dr. Kevin Schneider, for his valuable insights and guidance. I am additionally grateful to Dr. Chanchal Roy for his helpful review and feedback.

REFERENCES

[1] Ajmain I Alam, Palash R Roy, Farouq Al-Omari, Chanchal K Roy, Banani Roy, and Kevin A Schneider. Gptclonebench: A comprehensive benchmark of semantic clones and cross-language clones using gpt-3 model and semanticclonebench. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 1–13. IEEE, 2023.

[2] Ajmain Inqiad Alam. Application of large language models in code clone: Benchmarking, evaluation and detection, 2024.

[3] Sarveshwar Bharti and Hardeep Singh. Proactively managing clones inside an ide: a systematic literature review. *International Journal of Computers and Applications*, 44(3):230–249, 2022.

[4] Louis-François Bouchard and Louie Peters. *Building LLMs for production: enhancing LLM abilities and reliability with prompting, fine-tuning, and RAG*. Towards AI, Inc., 2024.

[5] Julius Davies, Daniel M German, Michael W Godfrey, and Abram Hindle. Software bertillonage: Finding the provenance of an entity. In *Proceedings of the 8th working conference on mining software repositories*, pages 183–192, 2011.

[6] Farima Farmahinifarahani, Petr Babkin, Salwa Alamir, and Xiaomo Liu. From zero to sixty at the speed of rag: Improving yaml recipe generation via retrieval. In *2025 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code)*, pages 49–56. IEEE, 2025.

[7] Marios Fokaefs, Nikolaos Tsantalis, Eleni Stroulia, and Alexander Chatzigeorgiou. Jdeodorant: identification and application of extract class refactorings. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 1037–1039, 2011.

[8] Martin Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[9] GitClear. Ai copilot code quality: 2025 look back at 12 months of data. Technical report, GitClear, 2025. Accessed: 2025-04-26.

[10] GitHub Blog. Survey reveals ai's impact on the developer experience, 2023.

[11] Armijn Hemel, Karl Trygve Kalleberg, Rob Vermaas, and Eelco Dolstra. Finding software license violations through binary code clone detection. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 63–72, 2011.

[12] Badri Narayanan K, Sreeja Nukarapu, Devatha Krishna Sai, and Bharath Reddy Gudibandi. A refactoring advisor for enhanced cloned software: Based on several machine learning techniques. EAI, 8 2024.

[13] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[14] Junyi Li, Jie Chen, Ruiyang Ren, Xiaoxue Cheng, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. The dawn after the dark: An empirical study on factuality hallucination in large language models. *arXiv preprint arXiv:2401.03205*, 2024.

[15] Manishankar Mandal, Chanchal K Roy, and Kevin A Schneider. Automatic ranking of clones for refactoring through mining association rules. In *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pages 114–123. IEEE, 2014.

[16] Manishankar Mondal, Chanchal K Roy, and Kevin A Schneider. A survey on clone refactoring and tracking. *Journal of Systems and Software*, 159:110429, 2020.

[17] Ronak Pradeep, Nandan Thakur, Shivani Upadhyay, Daniel Campos, Nick Craswell, Ian Soboroff, Hoa Trang Dang, and Jimmy Lin. The great nugget recall: Automating fact extraction and rag evaluation with large language models. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 180–190, 2025.

[18] Chanchal K Roy, Minhaz F Zibran, and Rainer Koschke. The vision of software clone management: Past, present, and future (keynote paper). In *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pages 18–33. IEEE, 2014.

[19] Chanchal Kumar Roy and James R Cordy. A survey on software clone detection research. *Queen's School of computing TR*, 541(115):64–68, 2007.

[20] Palash R Roy, Ajmain I Alam, Farouq Al-Omari, Banani Roy, Chanchal K Roy, and Kevin A Schneider. Unveiling the potential of large language models in generating semantic and cross-language clones. In *2023 IEEE 17th International Workshop on Software Clones (IWSC)*, pages 22–28. IEEE, 2023.

[21] Palash Ranjan Roy. An exploratory study on the roles of llms in code similarity detection and generation, 2024.

[22] Vaibhav Saini, Farima Farmahinifarahani, Hitesh Sajnani, and Cristina Lopes. Oreo: Scaling clone detection beyond near-miss clones. In *Code Clone Analysis: Research, Tools, and Practices*, pages 63–74. Springer, 2021.

[23] Abdullah M Sheneamer. An automatic advisor for refactoring software clones based on machine learning. *IEEE Access*, 8:124978–124988, 2020.

[24] Nikolaos Tsantalis, Theodoros Chaikalis, and Alexander Chatzigeorgiou. Jdeodorant: Identification and removal of type-checking bad smells. In *2008 12th European conference on software maintenance and reengineering*, pages 329–331. IEEE, 2008.

[25] Ruotong Wang, Ruijia Cheng, Denae Ford, and Thomas Zimmermann. Investigating and designing for trust in ai-powered code generation tools. In *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*, pages 1475–1493, 2024.

[26] Christopher D Wirz, Julie L Demuth, Ann Bostrom, Mariana G Cains, Imme Ebert-Uphoff, David John Gagne II, Andrea Schumacher, Amy McGovern, and Deianna Madlambayan. (re) conceptualizing trustworthy ai: A foundation for change. *Artificial Intelligence*, page 104309, 2025.

[27] Weiwei Xu, Kai Gao, Hao He, and Minghui Zhou. Licoeval: Evaluating llms on license compliance in code generation. *arXiv preprint arXiv:2408.02487*, 2024.

[28] Ruru Yue, Zhe Gao, Na Meng, Yingfei Xiong, Xiaoyin Wang, and J David Morgenthaler. Automatic clone recommendation for refactoring based on the present and the past. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 115–126. IEEE, 2018.

[29] Gang Zhang, Xin Peng, Zhenchang Xing, Shihai Jiang, Hai Wang, and Wenyun Zhao. Towards contextual and on-demand code clone management by continuous monitoring. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 497–507. IEEE, 2013.

[30] Gang Zhang, Xin Peng, Zhenchang Xing, and Wenyun Zhao. Cloning practices: Why developers clone and what can be changed. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 285–294. IEEE, 2012.