An Empirical Study of Code Clone Genealogies in Human–AI Collaborative Development

Denis Sousa
State University of Ceará
Fortaleza, Brazil
denis.sousa@aluno.uece.br

Italo Uchoa
State University of Ceará
Fortaleza, Brazil
italo.uchoa@aluno.uece.br

Matheus Paixao
State University of Ceará
Fortaleza, Brazil
matheus.paixao@uece.br

Chaiyong Ragkhitwetsagul
Faculty of ICT, Mahidol University
Nakhon Pathom, Thailand
chaiyong.rag@mahidol.ac.th

Thiago Lima
State University of Ceará
Fortaleza, Brazil
thiaguinho.lima@aluno.uece.br

Abstract

Code clones consist of two or more identical or similar code snippets. Code clones hurt maintainability by requiring synchronized updates across multiple locations and increasing the risk of inconsistent changes. To understand how clones evolve, the genealogy of code clones captures the evolutionary history of duplicated code snippets by linking them across successive versions of the software system. Since the emergence of Large Language Models (LLMs), software engineering has been reshaped, with code written and evolved differently. This evolution has given rise to coding agents who act as partners to developers. While code clone genealogy is well understood in human-centric development, its evolution in human–agent collaborative projects remains unclear. In this study, we analyze 350 code clone lineages across 6 software projects in which human actively colaborate with coding agents. We observed that humans introduce 85.71% of code clones, whereas agents contribute only 14.29%. Despite similar clone survival rates for both humans (80%) and agents (76%), the maintenance dynamics differ significantly. The analysis of genealogies reveals that humans predominate in maintaining lineages created by agents. These findings highlight that humans remain critical for the evolution of code generated by coding agents.

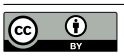
CCS Concepts

• **Software and its engineering → Software configuration management and version control systems**.

Keywords

Code Clone Genealogy, Collaborative Development, Coding Agents

ACM Reference Format:
Denis Sousa, Italo Uchoa, Matheus Paixao, Chaiyong Ragkhitwetsagul, and Thiago Lima. 2026. An Empirical Study of Code Clone Genealogies in Human–AI Collaborative Development. In *23rd International Conference on Mining Software Repositories (MSR '26), April 13–14, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3793302.3793613



MSR '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2474-9/2026/04
https://doi.org/10.1145/3793302.3793613

1 Introduction

Code clones are two or more code snippets that are identical or sufficiently similar in structure, syntax, or semantics [1, 22, 25]. Clones typically arise when developers copy and reuse existing code, either within the same codebase or across different codebases. As a result, duplication often requires developers to replicate changes in multiple places, which can degrade the code structure and increase the risk of inconsistent updates [4, 10].

Several tools and techniques have been proposed for code clone detection [1, 19, 22]. They further enable analyses of how clones are introduced, modified, and removed over time [16, 26]. In this context, the genealogy of code clones captures the evolutionary history of duplicated snippets by linking them across successive versions of the software system [12, 20]. When comparing two consecutive versions of a clone, it is necessary to identify the occurrence of *Change Patterns* and *Evolution Patterns*, following prior genealogy work [12, 20].

A *Change Pattern* occurs whenever at least one code snippet within a clone is modified between two versions. We distinguish three types of Change Patterns: *Consistent*, when all snippets in the clone undergo equivalent modifications and remain clones of one another; *Inconsistent*, when only a subset of snippets is modified so that the clone splits into different groups; and *Same*, when no snippet is changed. An *Evolution Pattern* occurs whenever the number of snippets in a clone changes between two versions. We distinguish three types of Evolution Patterns: *Addition*, when new snippets are added to the clone; *Subtraction*, when some snippets are removed; and *Same*, when no snippets are added or removed.

Since the emergence of Large Language Models (LLMs), software engineering has been reshaped in how code is written and maintained [13, 15]. This evolution has given rise to coding agents who act as partners to developers [9]. These agents have transitioned from passive auto-complete tools to active collaborators capable of autonomously generating pull requests (PRs), fixing bugs, and implementing features alongside human developers [18]. Collaborative projects involving human developers and coding agents are increasingly common, largely driven by productivity gains [14, 17]. A challenge that arises in this context is the management of code clones. It is unclear whether agents introduce original or reused code and how they and humans modify and interact with each other's clones.

In this study, we investigate code clone genealogies in collaborative software projects developed jointly by human developers and coding agents. Leveraging the AIDev dataset [14], we analyze six collaborative development projects between humans and agents. We extract code clone genealogies using OmniCCG [2], which leverages an embedded clone detector configured to use NiCad [5]. We identify 350 clone lineages and find that humans introduce the vast majority of them, accounting for 85.71% of lineages whose first observed cloned instance appears in a human-merged PR, compared to 14.29% for agent-merged PRs. Survival rates are similar, standing at 80% for human-origin lineages and 76% for agent-origin ones. When analyzing the *Change Patterns* and *Evolution Patterns*, we observe that human-origin lineages exhibit a balanced mix of human and agent maintenance, whereas humans predominate in maintaining lineages created by agents.

Our insights reveal that while agents are capable contributors, humans remain actively engaged in the evolution and maintenance of agent-generated code clones. To the best of our knowledge, this is the first empirical study on code clone genealogy in collaborative human–agent software development. All data and scripts are available in our replication package [23].

2 Background

This section presents the main concepts related to code clones and code clone genealogy as discussed in the literature [1, 11, 12, 22, 25].

Code clones consist of two or more code snippets that are identical or similar to each other [1, 22, 25]. To identify code clones in software systems, several tools and techniques have been developed to support their detection [1, 5, 19, 21, 22]. In this paper, we use NiCad [5] to detect code clones because it is widely used in code clone genealogy studies [3, 6, 8, 12, 24].

Code clone genealogy captures the evolutionary history of duplicated code snippets across successive software versions [11, 12, 20]. In line with prior work, we describe genealogy in terms of *Change Patterns* and *Evolution Patterns*, as defined in Section 1. Change Patterns characterize how the content of snippets within a clone changes between versions, while Evolution Patterns characterize how the number of snippets changes [12, 20].

Clone lineage is an ordered sequence of all versions of a code clone across the system's history [12, 20]. Following Kim et al. [12], a clone in system version n is linked to its counterpart in version n–1 by a Change Pattern and an Evolution Pattern. ***Clone genealogy*** is a set of clone lineages that originate from the same code clone [12, 20]. After diverging from their common origin, lineages may exhibit different Change Patterns and Evolution Patterns.

Figure 1 shows an example of a clone genealogy composed of two lineages. Code snippets A and B represent a clone in the initial system version (v1) and constitute the origin of the genealogy. In v2, snippets A and B remain unchanged, and snippets C and D are added, corresponding to a *Same* Change Pattern and an *Addition* Evolution Pattern. In v3, only snippets C and D are modified equivalently, corresponding to an *Inconsistent* Change Pattern. In addition, no snippets are added or removed, indicating a *Same* Evolution Pattern. The *Inconsistent* pattern denotes a point of lineage diversion, yielding a first lineage composed of snippets A and B and a second lineage composed of snippets C and D. In v4, within the first

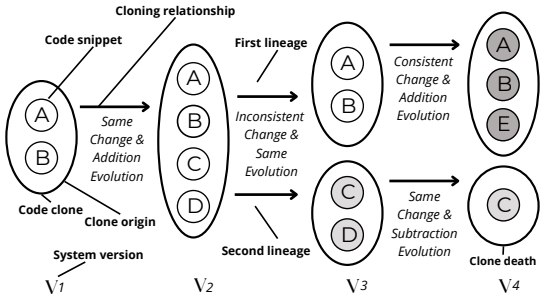


Figure 1: Example of a code clone genealogy.

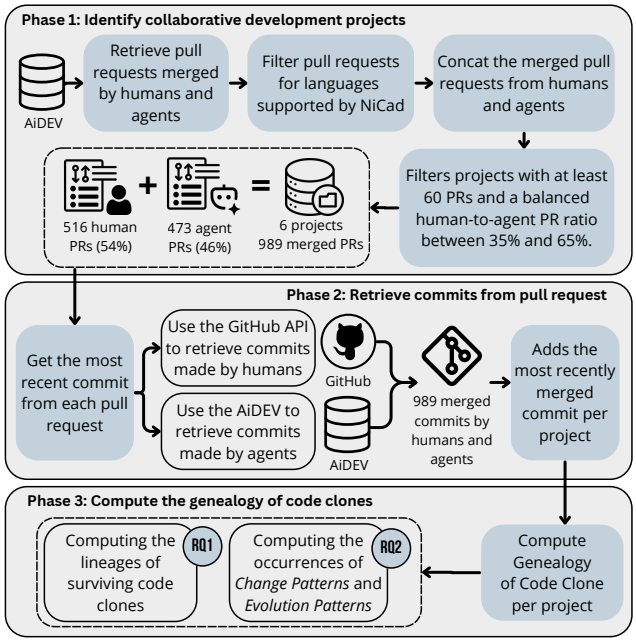


Figure 2: Methodology used in this study.

lineage, snippets A and B are modified equivalently and snippet E is added, corresponding to a *Consistent* Change Pattern and an *Addition* Evolution Pattern. Within the second lineage, snippet D is removed while snippet C remains unchanged, corresponding to a *Same* Change Pattern and a *Subtraction* Evolution Pattern. After this removal, snippet C no longer forms a clone with any other snippet, marking the death of the clone and its lineage.

3 Empirical Methodology

In this study, we analyze the genealogy of code clones in collaborative projects developed jointly by humans and coding agents. We address the following research questions:

- *RQ1: To what extent do clone lineages originating from humans and agents survive?*
- *RQ2: To what extent do humans and agents differ in their contributions to clone lineages?*

To address these RQs, we conducted a multi-phase process, as summarized in Figure 2.

Table 1: Summary of projects selected in Phase 1.

Project	Language	PRs	Humans	Agents	Human%	Agent%
airbyte	Python	259	150	109	57.92%	42.08%
marimo	Python	75	44	31	58.67%	41.33%
mlflow	Python	130	55	75	42.31%	57.69%
aspire	C#	182	77	105	42.31%	57.69%
testfx	C#	62	27	35	43.55%	56.45%
flexile	Ruby	281	120	161	42.70%	57.30%
Total	-	**989**	**473**	**516**	**47.83%**	**52.17%**

3.1 Phase 1: Identify Collaborative Projects

This phase aims to construct a dataset of projects in which humans and agents both contribute in a balanced manner. We begin by retrieving PRs merged by humans and by agents from AIDev, using the `pull_request` and `human_pull_request` tables. Next, we filter PRs to retain only those written in programming languages supported by NiCad (C, C#, Java, PHP, Ruby, and Python). In total, the dataset comprises 25,904 merged PRs, including 24,014 merged by coding agents and 1,890 merged by human developers.

Next, we apply repository-level inclusion criteria to ensure sufficient activity and balanced collaboration. We retain projects with at least 60 merged PRs and with an agent participation rate between 35% and 65% of merged PRs. Applying these criteria yields six projects comprising 989 merged PRs in total, of which 516 (54%) were merged by humans, and 473 (46%) were merged by agents. Table 1 details the resulting set of balanced projects and reports, for each project, the total number of merged PRs as well as the corresponding human and agent shares. Overall, the selected projects range from 62 to 281 merged PRs, and all satisfy the targeted balance, with agent participation varying from 41.33% to 57.69%. This set of projects forms the basis for all subsequent analyses.

3.2 Phase 2: Retrieve Commits from PRs

In this phase we retrieve the commits from PRs merged by humans and agents. Since a PR contains multiple commits, we select the last commit made in the PR. For PRs created by humans, we use the GitHub API[1] to obtain the commit. For PRs created by agents, we take the most recent commit in AiDEV using the `pr_commits` table.

At the end of this phase, we obtain 989 merged commits (one per PR), each labeled by contributor type (human or agent). These commits are then ordered chronologically to form the sequence of "consecutive versions" for genealogy construction in Phase 3.

We consider only these 989 snapshots, ignoring any intermediate commits between merged PRs. This results in partial genealogies, as clones may have been introduced or modified in commits outside this sequence.

3.3 Phase 3: Construct Code Clone Genealogy

This phase computes clone genealogies per project and derives code-clone genealogy-based measures to answer RQ1 and RQ2. We applied OmniCCG embedded with NiCad [5], configured for **function-level clone detection** with a minimum clone size of 6 lines and a 30% dissimilarity threshold (equivalent to 70% similarity), on the project snapshots corresponding to the commits extracted in Phase 2. These are standard parameters in prior code clone genealogy studies [12, 20, 24], balancing precision and recall for tracking clone evolution (see replication package for full configuration).

[1] https://api.github.com

Table 2: Clone lineage survival by origin.

Project	Lineages	Human		Agent		Overall		Total PRs
		Alive	Dead	Alive	Dead	Alive	Dead	
airbyte	65	55	6	1	3	56 (86.2%)	9 (13.8%)	259
aspire	38	25	12	1	0	26 (68.4%)	12 (31.6%)	182
flexile	48	40	6	1	1	41 (85.4%)	7 (14.6%)	281
marimo	50	12	1	30	7	42 (84.0%)	8 (16.0%)	75
mlflow	119	82	32	4	1	86 (72.3%)	33 (27.7%)	130
testfx	30	26	3	1	0	27 (90.0%)	3 (10.0%)	62
Total	**350**	**240**	**60**	**38**	**12**	**278 (79.4%)**	**72 (20.6%)**	**989**

Using the detected clones, we construct clone genealogies by linking clone instances across consecutive commits (the 989 snapshots ordered chronologically from Phase 2), thereby forming clone lineages. From the resulting genealogies, we perform two complementary analyses aligned with our research questions.

3.3.1 Answering RQ1. We quantify the number of introduced by both humans and agents, where a lineage's origin is defined by the contributor type (human or agent) of the merged PR in which the clone pair first appears in our sequence of 989 commits. We identify lineage survival by classifying lineages as *alive* if they are present in the latest merge commit and *dead* if they are no longer present. Finally, we compute lineage longevity per project.

3.3.2 Answering RQ2. For all lineages introduced by both humans and agents, we count the consistent and inconsistent snippet changes (*Change Patterns*) performed by humans and agents. Furthermore, we count the additions and subtractions of snippets (*Evolution Patterns*) performed by humans and agents.

4 Results

4.1 RQ1: To what extent do clone lineages originating from humans and agents differ?

Table 2 details the clone lineages across the six analyzed repositories, encompassing a total of 350 lineages identified within the merged PRs from AIDev. The vast majority of these lineages originate from human developers, with 300 lineages (85.71%), while coding agents contributed 50 lineages (14.29%). Overall, the stability of code clones in collaborative human-agent projects is high, with 278 lineages (79.43%) remaining alive at the end of the observation period, while only 72 (20.57%) were discontinued.

When examining survival rates by origin, we observe a slight divergence. Lineages originating from human developers exhibit a survival rate of 80% (60 discontinued). In comparison, lineages originating from coding agents exhibit a survival rate of 76% (12 discontinued). While lineages originating from humans appear slightly more persistent, the substantial 76% survival rate for clones introduced by agents indicates that agent-generated code is largely accepted. The *marimo* project accounts for the majority of lineages introduced by agents (74%), whereas the clone genealigies of projects like *airbyte* and *mlflow* are predominantly human-driven, despite the presence of sporadic agent contributions.

Table 3: Clone lineage evolution operations

Project	Origin	Change patterns			Evolution patterns		
		Human	Agent	Total	Human	Agent	Total
airbyte	Human	22 (46.8%)	25 (53.2%)	47	49 (38.9%)	77 (61.1%)	126
	Agent	1 (100%)	0 (0%)	1	0 (0%)	0 (0%)	0
aspire	Human	3 (13.0%)	20 (87.0%)	23	4 (100%)	0 (0%)	4
	Agent	1 (100%)	0 (0%)	1	1 (100%)	0 (0%)	1
flexile	Human	2 (16.7%)	10 (83.3%)	12	5 (20.0%)	20 (80.0%)	25
	Agent	0 (0%)	0 (0%)	0	0 (0%)	0 (0%)	0
marimo	Human	2 (100%)	0 (0%)	2	1 (100%)	0 (0%)	1
	Agent	30 (90.9%)	3 (9.1%)	33	5 (83.3%)	1 (16.7%)	6
mlflow	Human	80 (77.7%)	23 (22.3%)	103	21 (80.8%)	5 (19.2%)	26
	Agent	1 (50.0%)	1 (50.0%)	2	1 (25.0%)	3 (75.0%)	4
testfx	Human	16 (84.2%)	3 (15.8%)	19	3 (60.0%)	2 (40.0%)	5
	Agent	0 (0%)	0 (0%)	0	0 (0%)	0 (0%)	0
Total (Human-Origin)		125 (60.7%)	81 (39.3%)	206	83 (44.4%)	104 (55.6%)	187
Total (Agent-Origin)		33 (89.2%)	4 (10.8%)	37	7 (63.6%)	4 (36.4%)	11
Grand Total		**158 (65.0%)**	**85 (35.0%)**	**243**	**90 (45.5%)**	**108 (54.5%)**	**198**

Answer to RQ1: Humans introduce more clone lineages than agents, with survival rates of 80% and 76%, respectively.

4.2 RQ2: To what extent do humans and agents differ in their contributions to clone lineages?

Table 3 summarizes human and agent contributions to clone lineages by project and lineage origin, distinguishing *Change Patterns* and *Evolution Patterns* over time.

In lineages originally created by humans, we found 206 *Change Pattern* occurrences. Humans remain the primary maintainers of their own code clones, accounting for 60.68% of these operations. However, agents contribute significantly (39.32%), particularly in *aspire* (87%) and *flexile* (83%) projects, where they actively modify human code. Regarding the nature of these operations, humans and agents lean towards *Inconsistent* patterns. Specifically, human changes are 44% *Consistent* and 56% *Inconsistent*, while agent changes are 44.4% *Consistent* and 55% *Inconsistent*. We found 187 *Evolution Pattern* occurrences. Agents account for the majority of these operations, representing 55.61% of the total. This trend is heavily driven by *airbyte* (61.1%) and *flexile* (80.0%) projects, where agents actively perform the majority of *Evolution Patterns*. Regarding the remaining projects, specifically *mlflow*, *aspire*, and *marimo*, humans still dominate the *Evolution Patterns* of the lineages.

In agent-created lineages, we found 37 *Change Pattern* occurrences. Humans account for the majority of these operations, representing 89% of the total. This trend is evident in the *marimo* project (90.9%), where humans actively revise clones generated by agents. We found 11 *Evolution Pattern* occurrences. Humans account for the majority of these operations, representing 63.64% of the total. Activity here is sparse, occurring primarily in *marimo* and *mlflow*, while *airbyte* shows no activity. This suggests that agent-created clones in the remaining projects are either short-lived or static.

Overall, the results show a clear asymmetry. Agents frequently assist in maintaining human-created clone lineages, while agent-created lineages depend mostly on humans for maintenance.

RQ2: For *Change* and *Evolution* patterns, contributions to human-created lineages are balanced between humans and agents; for agent-created lineages, humans dominate maintenance.

5 Threats to the Validity

Internal Validity: our study relies on the accuracy of the tools employed for data extraction and analysis. We utilized OmniCCG [2] configured with NiCad [5] for clone detection. Although NiCad is a widely accepted state-of-the-art tool, clone detection errors may affect genealogy construction. We mitigate this by using standard configurations from prior clone genealogy studies [24]. Furthermore, the classification of contributors relies on the AIDev dataset. While AIDev provides a curated list of human and agent PRs, any misclassifications in the ground truth could affect our comparative analysis of lineage origins.

External Validity: a primary threat to the generalization of our results is the number of analyzed projects. By applying strict inclusion criteria that require a balanced share of merged PRs between 35% and 65% to enable fair comparisons of co-authorship dynamics, we narrowed the dataset to six projects. While this selection was necessary to observe genuine collaboration rather than isolated agent contributions, the findings may not apply to projects where agent adoption is sparse.

6 Related Work

Van et al. [24] use clone genealogy to compare how clones are introduced, modified, and removed in production versus test code. They build genealogies with NiCad [5] and iClones [7]. Ehsan et al. [6] rank code clones by fault risk using machine-learning models built on genealogy features. Their results show that clone age and the number of distinct developers are strong predictors of risky clones. Hu et al. [8] assess clone harmfulness by mining clone genealogies and deriving quantitative indicators from clone evolution to identify clones likely to increase maintenance cost and risk.

Overall, prior work leverages clone genealogy to study clone evolution for maintenance. In contrast, our study uses clone genealogy to compare how humans and coding agents introduce, modify, and remove clones in hybrid development.

7 Ethical Implications

The study analyzed only publicly available software repositories and collected or inferred no developers' personal data. We use the AiDEV [14] dataset, which labels PRs in public open-source GitHub repositories as authored by humans or agents, and the GitHub API to only obtain commits from human-authored PRs.

8 Conclusion

In this study, we analyzed the code clone genealogies in collaborative software projects developed jointly by human developers and coding agents. Across 6 projects (989 merged PRs), most clone lineages are introduced by humans (86.23%), and their survival is comparable to agent-introduced lineages, both around 81% remain alive. Humans perform most lineage maintenance, accounting for the majority of both *Change Patterns* and *Evolution Patterns*, while agent activity is smaller overall and concentrated in a few projects. Future work will examine which types of coding agents contribute most to clone genealogy evolution and whether human developers are aware of when agents introduce clones in their projects.

References

[1] Qurat Ul Ain, Wasi Haider Butt, Muhammad Waseem Anwar, Farooque Azam, and Bilal Maqbool. 2019. A systematic review on code clone detection. *IEEE access* 7 (2019), 86121–86144.

[2] Anonymous. 2026. OmniCCG: Agnostic Code Clone Genealogy Extractor. (2026). Manuscript under review. Submitted to Data and Tools track of the 23rd International Conference on Mining Software Repositories (MSR'26).

[3] Maram Assi, Safwat Hassan, and Ying Zou. 2024. Unraveling code clone dynamics in deep learning frameworks. *arXiv preprint arXiv:2404.17046* (2024).

[4] Nicolas Bettenburg, Weiyi Shang, Walid M Ibrahim, Bram Adams, Ying Zou, and Ahmed E Hassan. 2012. An empirical study on inconsistent changes to code clones at the release level. *Science of Computer Programming* 77, 6 (2012), 760–776.

[5] James R Cordy and Chanchal K Roy. 2011. The NiCad clone detector. In *2011 IEEE 19th international conference on program comprehension*. IEEE, 219–220.

[6] Osama Ehsan, Foutse Khomh, Ying Zou, and Dong Qiu. 2023. Ranking code clones to support maintenance activities. *Empirical Software Engineering* 28, 3 (2023), 70.

[7] Nils Göde and Rainer Koschke. 2009. Incremental clone detection. In *2009 13th European conference on software maintenance and reengineering*. IEEE, 219–228.

[8] Bin Hu, Yijian Wu, Xin Peng, Jun Sun, Nanjie Zhan, and Jun Wu. 2021. Assessing code clone harmfulness: Indicators, factors, and counter measures. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 225–236.

[9] Haolin Jin, Linghan Huang, Haipeng Cai, Jun Yan, Bo Li, and Huaming Chen. 2024. From llms to llm-based agents for software engineering: A survey of current, challenges and future. *arXiv preprint arXiv:2408.02479* (2024).

[10] Elmar Juergens, Benjamin Hummel, Florian Deissenboeck, and Martin Feilkas. 2008. Static bug detection through analysis of inconsistent clones. In *Software Engineering 2008*. Gesellschaft für Informatik e. V., 443–446.

[11] Miryung Kim and David Notkin. 2005. Using a clone genealogy extractor for understanding and supporting evolution of code clones. *ACM SIGSOFT Software Engineering Notes* 30, 4 (2005), 1–5.

[12] Miryung Kim, Vibha Sazawal, David Notkin, and Gail Murphy. 2005. An empirical study of code clone genealogies. In *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*. 187–196.

[13] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems* 35 (2022), 22199–22213.

[14] Hao Li, Haoxiang Zhang, and Ahmed E. Hassan. 2025. The Rise of AI Teammates in Software Engineering (SE) 3.0: How Autonomous Coding Agents Are Reshaping Software Engineering. *arXiv preprint arXiv:2507.15003* (2025).

[15] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[16] Jeremy R Pate, Robert Tairas, and Nicholas A Kraft. 2013. Clone evolution: a systematic review. *Journal of software: Evolution and Process* 25, 3 (2013), 261–283.

[17] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. 2023. The impact of ai on developer productivity: Evidence from github copilot. *arXiv preprint arXiv:2302.06590* (2023).

[18] Meghana Puvvadi, Sai Kumar Arava, Adarsh Santoria, Sesha Sai Prasanna Chennupati, and Harsha Vardhan Puvvadi. 2025. Coding agents: A comprehensive survey of automated bug fixing systems and benchmarks. In *2025 IEEE 14th International Conference on Communication Systems and Network Technologies (CSNT)*. IEEE, 680–686.

[19] Chaiyong Ragkhitwetsagul and Jens Krinke. 2019. Siamese: scalable and incremental code clone search via multiple code representations. *Empirical Software Engineering* 24, 4 (2019), 2236–2284.

[20] Ripon K Saha, Muhammad Asaduzzaman, Minhaz F Zibran, Chanchal K Roy, and Kevin A Schneider. 2010. Evaluating code clone genealogies at release level: An empirical study. In *2010 10th IEEE Working Conference on Source Code Analysis and Manipulation*. IEEE, 87–96.

[21] Hitesh Sajnani, Vaibhav Saini, Jeffrey Svajlenko, Chanchal K Roy, and Cristina V Lopes. 2016. Sourcerercc: Scaling code clone detection to big-code. In *Proceedings of the 38th international conference on software engineering*. 1157–1168.

[22] G Shobha, Ajay Rana, Vineet Kansal, and Sarvesh Tanwar. 2021. Code clone detection—a systematic review. *Emerging Technologies in Data Mining and Information Security: Proceedings of IEMIS 2020, Volume 2* (2021), 645–655.

[23] Sousa, Denis and Uchoa, Italo and Paixao, Matheus and Ragkhitwetsagul, Chaiyong and Lima, Thiago. 2026. Replication Package for the paper: "An Empirical Study of Code Clone Genealogies in Human–AI Collaborative Development". https://zenodo.org/records/18380268

[24] Brent van Bladel and Serge Demeyer. 2023. A comparative study of code clone genealogies in test code and production code. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 913–920.

[25] Morteza Zakeri-Nasrabadi, Saeed Parsa, Mohammad Ramezani, Chanchal Roy, and Masoud Ekhtiarzadeh. 2023. A systematic literature review on source code similarity measurement and clone detection: Techniques, applications, and challenges. *Journal of Systems and Software* 204 (2023), 111796.

[26] Yan Zhong, Xunhui Zhang, Wang Tao, and Yanzhi Zhang. 2022. A systematic literature review of clone evolution. In *Proceedings of the 5th International Conference on Computer Science and Software Engineering*. 461–473.