More Code, Less Reuse: Investigation on Code Quality and Reviewer Sentiment towards AI-generated Pull Requests

Haoming Huang*
Institute of Science Tokyo
Tokyo, Japan
haoming@se.comp.isct.ac.jp

Pongchai Jaisri*
Nara Institute of Science and Technology
Nara, Japan
jaisri.pongchai.js3@naist.ac.jp

Shota Shimizu
Ritsumeikan University
Ibaraki, Japan
is0618vf@ed.ritsumei.ac.jp

Lingfeng Chen
Kyushu University
Fukuoka, Japan
lingfeng@posl.ait.kyushu-u.ac.jp

Sota Nakashima
Kyushu University
Fukuoka, Japan
nakashima@posl.ait.kyushu-u.ac.jp

Gema Rodríguez-Pérez
Computer Science
University of British Columbia
Kelowna, BC, Canada
gema.rodriguezperez@ubc.ca

Abstract

Large Language Model (LLM) Agents are advancing quickly, with the increasing leveraging of LLM Agents to assist in development tasks such as code generation. While LLM Agents accelerate code generation, studies indicate they may introduce adverse effects on development. However, existing metrics solely measure pass rates, failing to reflect impacts on long-term maintainability and readability, and failing to capture human intuitive evaluations of PR. To increase the comprehensiveness of this problem, we investigate and evaluate the characteristics of LLM to know the pull requests' characteristics beyond the pass rate. We observe the code quality and maintainability within PRs based on code metrics to evaluate objective characteristics and developers' reactions to the pull requests from both humans and LLM's generation. Evaluation results indicate that LLM Agents frequently disregard code reuse opportunities, resulting in higher levels of redundancy compared to human developers. In contrast to the quality issues, our emotions analysis reveals that reviewers tend to express more neutral or positive emotions towards AI-generated contributions than human ones. This disconnect suggests that the surface-level plausibility of AI code masks redundancy, leading to the silent accumulation of technical debt in real-world development environments. Our research provides insights for improving human-AI collaboration.

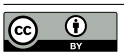
CCS Concepts

• **Software and its engineering → Software libraries and repositories**; *Maintaining software*; *Automatic programming*.

Keywords

Large Language Models, Agents, Code Quality, Code Clone, Code Generation, Reviewer Sentiment

*Equal contribution



MSR '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2474-9/26/04
https://doi.org/10.1145/3793302.3793622

ACM Reference Format:
Haoming Huang, Pongchai Jaisri, Shota Shimizu, Lingfeng Chen, Sota Nakashima, and Gema Rodríguez-Pérez. 2026. More Code, Less Reuse: Investigation on Code Quality and Reviewer Sentiment towards AI-generated Pull Requests. 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/10.1145/3793302.3793622

1 Introduction

The software engineering (SE) landscape is undergoing a paradigm shift towards SE 3.0, where Large Language Model (LLM) agents are evolving from passive assistants to autonomous teammates [13]. The AIDev dataset [13] captures the emergence of pull requests (PRs) submitted autonomously by AI Agents (Agentic-PRs). While LLM Agents accelerate code generation, researches [3, 17] indicate they may introduce adverse effects on development. Existing benchmark (like SWE-Bench [9]) solely measure pass rates, failing to reflect impacts on long-term maintainability and readability. While code redundancy or code clone is a well-known issue in SE [18], evaluating it in the context of AI agents remains unexplored.

A concern with AI-generated code is code clone [7], a.k.a code redundancy. In SE, code redundancy is a known bad smell that increases maintenance effort [7]. Due to LLM's probabilistic model-based nature [4, 21], we believe generative AI models tend to produce *Type-4* semantic clones [18, 22], unlike human developers who often copy and paste code (creating *Type-1* or *Type-2* clones [10]). These clones implement the same logic as existing functions but use different syntax or variable names. This behavior violates the SE principle of reuse. However, existing tools are not effective at detecting these *Type-4* semantic clones (see section 5).

To address this problem, we propose a comprehensive evaluation framework. We assess Agentic-PRs from two perspectives: internal code quality and external reviewer perception. For internal code quality, we measure traditional metrics including *Lines of Code (LOC)* and *Cyclomatic Complexity* [15]. Furthermore, we propose a new metric called the *Max Redundancy Score (MRS)* to measure semantic redundancy. We use code emebdding model to build an approach, to detect *Type-4* semantic clones. For external reviewer

perception, we investigate how human reviewers react to Agentic-PRs. We conduct a sentiment analysis on review comments to see if reviewers identify the quality issues in AI-generated code.

We analyze the AIDev dataset [13] to answer the following:

- **RQ1:** How do AI-generated code differ from human-generated code in terms of code quality?
- **RQ2:** How does reviewer sentiment differ between AI-generated and human-written PRs

This study provides the first empirical evidence of the disconnect between AI-generated code redundancy and human reviewer sentiment. By highlighting the risk of silent technical debt, we offer insights for improving human-AI collaboration.

2 Methodology

2.1 Dataset

We utilize the AIDev dataset [13] for our empirical study. To ensure the analysis focuses on mature projects, we filter the dataset to include only Python repositories with more than 500 stars, following the popularity standard mentioned in AIDev [13]. We separate the data into human-generated PRs (Human-PRs) and agent-generated PRs (Agentic-PRs). To balance the depth of semantic analysis with computational feasibility, we employ a two-level data strategy:

- **Dataset A** (Full Python Dataset): This dataset comprises 3,858 PRs across all filtered Python repositories. We use this broad dataset to analyze traditional code metrics (RQ1) and reviewer sentiment (RQ2).
- **Dataset B** (Core Case Study): This dataset consists of 617 PRs from the crewAI repository, the repository with the most PRs in AIDev. As the most active repository in AIDev with high agent activity, it serves as a representative case for our computationally intensive redundancy analysis (RQ1).

2.2 RQ1: How do AI-generated code differ from human-generated code in terms of code quality?

This question compares the code quality between handcraft changes and AI-generated changes. We employ a multi-dimensional evaluation strategy including both traditional static metrics and a novel code redundancy analysis.

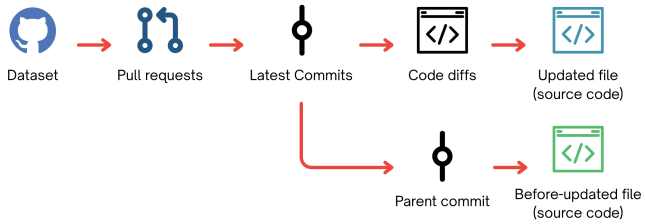


Figure 1: Overview of collecting changes from pull requests

As shown in Figure 1, We extracted pairs of source files representing the pre- and post-update states by identifying the modified files in each PR relative to their parent commits.

2.2.1 Traditional Metrics. We calculated *Cyclomatic Complexity (CC)* and raw metrics (*LOC*, *multi-lines-string*, *blank lines*) using Radon[1] for each file pair to compare human and AI contributions.

2.2.2 Redundancy Score. Due to the probabilistic nature of LLM models [4, 21], they are more prone to generating *Type-4* Clones, which reduce software maintainability. While LLMs can also produce *Type-1* and *Type-2* clones, existing tools that rely on literal keyword matching remain ineffective at identifying the more prevalent *Type-4* Clones.

To bridge this gap, we introduce a Snapshot-based Semantic Differential Analysis pipeline. We extract all existing functions from the snapshot (F_{base}) and generate semantic embeddings using CodeSage-Large [24], a model optimized for code representation. This allows us to capture functional equivalence beyond syntactic similarity. A naive comparison of added code yields false positives, as moving or renaming a function appears as an addition in Git diffs. To isolate true redundancy, we integrate PyRef [1], a refactoring detection tool, to identify and exclude *Move Method* and *Rename Method* instances from the set of new functions (F_{new}). We define the redundancy of a PR not by its average similarity, but by its worst offender. In maintenance, a single instance of severe duplication (e.g., rewriting a complex core utility) creates a single point of failure for future updates [10]. Thus, we calculate the *Max Redundancy Score (MRS)* by Equation 1;

$$\mathrm{MRS}(P_i) = \max_{f \in F_{new}^{(i)}} \left(\max_{g \in F_{base}^{(i)}} \cos(\mathbf{v}(f), \mathbf{v}(g)) \right) \tag{1}$$

where $\mathbf{v}(\cdot)$ denotes the embedding vector. We finally report the *Average Max Redundancy (AMR)* to compare human and AI groups.

2.3 RQ 2: How does reviewer sentiment differ between AI-generated and human-written PRs

To understand how human developers perceive code generated by AI agents, we analyze the sentiments expressed in code review comments. Specifically, we compare the emotional reactions in Agentic-PRs versus Human-PRs using Dataset A (the full Python dataset) for this analysis.

To obtain the sentiments, we use state-of-the-art sentiment analysis model, Emotion English DistilRoBERTa-base [8]. This model classifies emotions to Ekman's six basic emotions (anger, disgust, fear, joy, sadness, surprise) and neutral. Since the model has a maximum token length of 512, we exclude the data which was more than maximum token lengths.

To compare the emotional reactions in Agentic-PRs versus Human-PRs, we analyze in the following process. First, we used code review comments from *pr_comments*, *pr_review_comments_v2*, *pr_reviews* from AIdev [13]. To focus on developer sentiments, we exclude comments made by bots. Second, we filter Dataset A to identify the PRs made by Agents and Humans. Next, we classify the sentiments of comments made by each PR using Emotion English DistilRoBERTa-base. We calculate the average sentiment scores

[1] https://pypi.org/project/radon/

per PR to obtain how developers perceived within each PR. This process was applied separately to Agentic-PRs and Human-PRs.

Additionally, we identify the PRs with the highest sentiment scores for each emotion in both Agentic-PRs and Human-PRs. Analyzing these PRs provides further insights into how the two groups differ in terms of sentiments expressed during code reviews.

3 Results

3.1 Results of RQ1

Raw Code Metrics: As shown in Table 1, while code additions vary slightly between groups, human developers remove significantly ($p < 0.001$) more *multi-lines-string* (26.26) than AI agents (8.47), alongside a higher average *LOC* removal (25.51 vs. 16.60).

Table 1: Average Addition Lines Differences by Metric.

Change Types	PRs' types	LOC	Multi-Lines	Blank Lines
Addition	Human	23.01	15.79	6.16
	Agent	23.78	16.48	6.02
	Δ Diff	-0.77***	-0.69	0.14*
Removal	Human	25.51	26.26	6.99
	Agent	16.60	8.47	5.05
	Δ Diff	8.91	17.79***	1.94

* Significant ($p < 0.05$) *** Highly Significant ($p < 0.001$)

Cyclomatic Complexity (CC):. As shown in Table 2, the computation of the CC through all pairs, most of code changes or 18,968 pairs (85.02%) have zero complexity score. The result shows that only 3,333 from 22,311 pairs (14.94%) have scored more than 0 which include 1,343 pairs (6.02%) are low risk code removal pairs, 1,979 pairs (8.87%) are low risk code addition pairs and 11 pairs (0.05%) are high risk code addition pairs.

Table 2: Distribution of Complexity Score of Code Changes

PRs' types	Complexity Score (CC)		
	0 No Risk	0 < |CC| < 10 Low Risk	10+ High Risk
Human	8,643	1,463	6
Agent	10,325	1,859	5
All (PRs)	18,968	3,322	11

Redundancy: As shown in Figure 2, Agentic-PRs demonstrate a significantly higher level of redundancy. For low *Max Redundancy Score* (*MRS*, see subsubsection 2.2.2) sections, the distributions of AI agents and humans are largely similar. However, for medium *MRS* sections (0.3–0.8), AI agents exhibit higher *AMR*. We infer that both humans and AI agents produce some similar code (e.g., API calls or common patterns). However, LLM-generated code significantly exceeds human-written code in higher AMR ranges. Quantitatively, the *Average Max Redundancy (AMR)* for AI agents is 0.2867, compared to only *0.1532* for humans, representing a nearly *1.87x increase*. Mann-Whitney test confirms that this difference is statistically significant ($p < 0.001$).

Answer to RQ1: While traditional metrics show minimal differences between agentic-PRs and human-PRs, redundancy metric analysis shows code in *agentic-PRs contain significantly more redundancy* ($p < 0.001$).

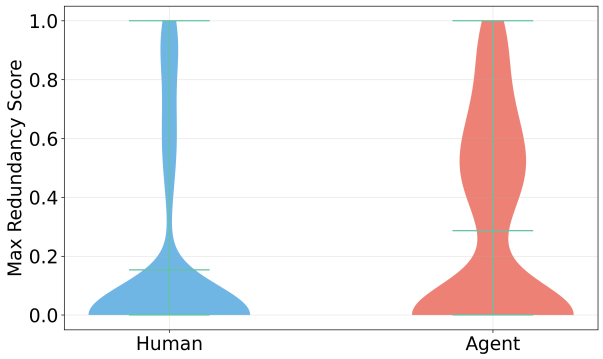


Figure 2: MRS Distribution Between Humans and AI

3.2 Results of RQ2

The result of RQ2 is shown in Figure 3. The most frequent sentiment is neutral, meaning that reviewers mostly review PRs respectfully for both Human-PRs and Agentic-PRs. However, comparing the differences between each sentiment, Agentic-PRs had a higher proportion of neutral, joy, and sadness, while Human-PRs had a higher proportion of disgust, anger, fear, and surprise.

Answer to RQ2: Our analysis shows that reviewers tend to express more *neutral or positive* emotions towards AI-generated contributions than human ones.

4 Discussion

4.1 Disconnect between Quality and Sentiment

Despite the similarity in traditional metrics, we found a more critical issue on AI-generated PR. Our redundancy analysis (RQ1) shows that AI agents introduce significantly more redundancy than humans. AMR for Agentic-PRs is nearly double that of Human-PRs. This confirms that AI agents tend to generate redundant code instead of reusing existing code.

However, RQ2 shows that reviewers do not react negatively to this issue. In fact, reviewers express fewer negative emotions (such as anger or disgust) towards Agentic-PRs compared to Human-PRs. This counter-intuitive disconnect may stem from the training objectives of current LLMs. Models aligned using Reinforcement Learning from Human Feedback[16] are optimized to maximize human preference and perceived helpfulness [5]. Existing studies observed that LLMs tend to prioritize sycophancy over maximizing the accuracy of truthful statements or code quality when responding[6, 14, 16, 17, 20, 23]. Consequently, AI agents tend to produce code that appears superficially correct and helpful.

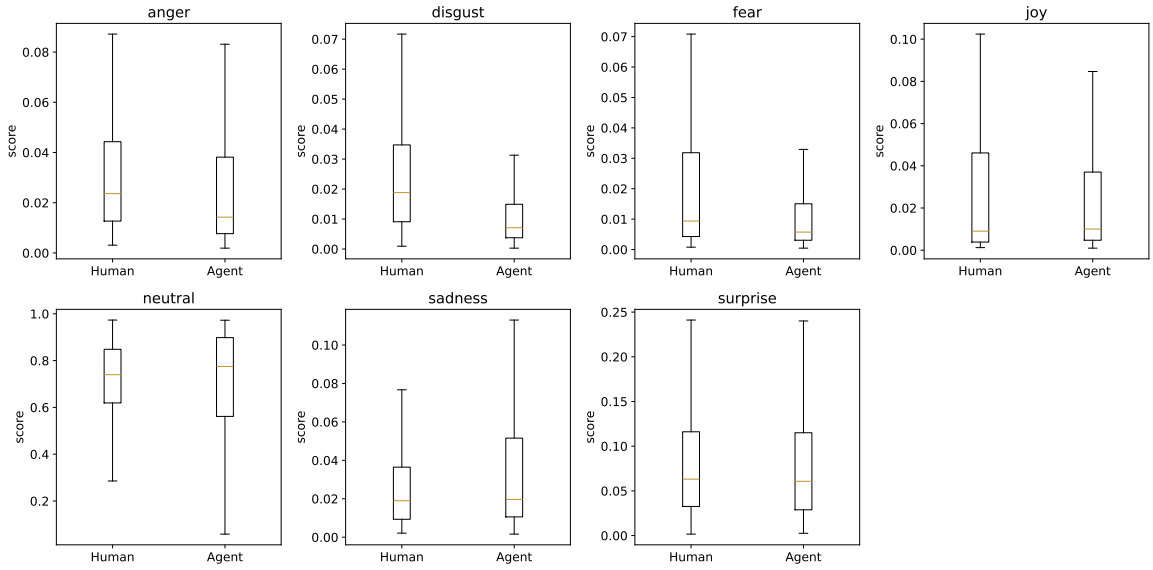


Figure 3: The reviewer sentiment score at Human-PRs and Agentic-PRs

This disconnect suggests a reviewer's blind spot. AI-generated code is syntactically correct and often passes tests. Because LLM agents generate code based on probability, the output looks plausible on the surface. As a result, reviewers may lower their vigilance. They focus on whether the code works (pass rate), but they fail to check if the code duplicates existing logic in the repository.

4.2 The Risk of Silent Technical Debt

This situation leads to the accumulation of silent technical debt. When an AI agent introduces a Type-4 clone, it creates a duplicate maintenance point. Previous studies have demonstrated that inconsistent updates to duplicated code are a primary source of software defects [10]. If a bug exists in the logic, a developer might fix the original utility function but miss the redundant copy created by the AI. Because reviewers are reacting positively (or neutrally) to these PRs, this debt accumulates unnoticed. Consequently, the codebase becomes bloated with redundant logic, which decreases long-term readability and increases future maintenance effort [18].

4.3 Implications

For Agent Builders: Researchers and tool developers need to address the issue of code redundancy. Our results show that current agents often ignore existing code. Therefore, builders should evaluate agents not just on pass rates, but also on code reuse metrics.
For Software Developers: Although AI agents increase coding speed in the short term, developers must be aware of the impact on long-term maintainability. The plausible code from AI may contain hidden redundancy. Reviewers should actively check for duplicated logic during code reviews. Ignoring this now may lead to higher review costs and maintenance efforts in the future.

5 Related Work

There are some existing approach for detect redundancy in the code (i.e., code clone), such as [2, 11]. Due to LLM's probabilistic model-based nature [4, 21], LLM models typically do not generate textually identical code clones. Instead, they tend to produce more Type-4 clones [18], redundant code that exhibits textual inconsistencies but shares similar semantic meaning. However, existing tools are not effective at detecting these Type-4 semantic clones in Agentic-PRs. Early approaches, such as [19], detect clones by comparing file hashes. This approach is too coarse because agents often add specific functions rather than duplicating entire files. Later approaches, such as [12], use information retrieval techniques to find similar code, but they rely on keyword matching and often miss semantic similarities when the syntax is different.

6 Threats to Validity

Internal Validity We relied on PyRef to filter out refactorings (Move Method/Rename Method). If PyRef fails to detect a refactoring, we might incorrectly label valid code movement as redundancy. To mitigate this, we manually verified 10 sample of the results. Additionally, we only analyzed the crewAI repository for RQ1 due to computational costs. While this repository is highly active, it may not represent all coding styles and results might not generalize.

External Validity We focused only on Python repositories. The behavior of AI agents may differ in other languages like Java or C++. Furthermore, our sentiment analysis model is trained on general English text. It may misinterpret technical discussions in code reviews as neutral or negative emotions.

7 Conclusion

This study investigates the impact of AI coding agents in SE beyond simple pass rates. We combined a semantic redundancy analysis

with a reviewer sentiment analysis. We found that AI agents produce more redundant code (Type-4 clones) than human developers. However, human reviewers do not punish this behavior; instead, they show less negative sentiment towards AI-generated PRs. We conclude that the surface-level plausibility of AI code can mask poor design choices. This leads to hidden technical debt, where redundancy accumulates without being noticed. As the use of AI agents increases, developers need to maintain high review standards to ensure long-term code maintainability.

Acknowledgments

This work was partly developed through discussions held during the AI-Driven SE Summit 2025.

References

[1] Hassan Atwi, Bin Lin, Nikolaos Tsantalis, Yutaro Kashiwa, Yasutaka Kamei, Naoyasu Ubayashi, Gabriele Bavota, and Michele Lanza. 2021. PYREF: 21st IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2021. (2021), 136–141. doi:10.1109/SCAM52516.2021.00025

[2] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. 1998. Clone Detection Using Abstract Syntax Trees. In *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*. 368–377. doi:10.1109/ICSM.1998.738528

[3] Joel Becker, Nate Rush, Elizabeth Barnes, and David Rein. 2025. Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity. arXiv:2507.09089 [cs] doi:10.48550/arXiv.2507.09089

[4] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A Neural Probabilistic Language Model. *Journal of machine learning research* 3, Feb (2003), 1137–1155.

[5] Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, Jérémy Scheurer, Javier Rando, Rachel Freedman, Tomek Korbak, David Lindner, Pedro Freire, Tony Tong Wang, Samuel Marks, Charbel-Raphael Segerie, Micah Carroll, Andi Peng, Phillip J. K. Christoffersen, Mehul Damani, Stewart Slocum, Usman Anwar, Anand Siththaranjan, Max Nadeau, Eric J. Michaud, Jacob Pfau, Dmitrii Krasheninnikov, Xin Chen, Lauro Langosco, Peter Hase, Erdem Biyik, Anca Dragan, David Krueger, Dorsa Sadigh, and Dylan Hadfield-Menell. 2023. Open Problems and Fundamental Limitations of Reinforcement Learning from Human Feedback. *Transactions on Machine Learning Research* (Sept. 2023).

[6] Aaron Fanous, Jacob Goldberg, Ank A. Agarwal, Joanna Lin, Anson Zhou, Roxana Daneshjou, and Sanmi Koyejo. 2025. SycEval: Evaluating LLM Sycophancy. arXiv:2502.08177 [cs] doi:10.48550/arXiv.2502.08177

[7] Martin Fowler. 2018. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Professional.

[8] Jochen Hartmann. 2022. Emotion English DistilRoBERTa-base. https://huggingface.co/j-hartmann/emotion-english-distilroberta-base/.

[9] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2023. SWE-bench: Can Language Models Resolve Real-world Github Issues?. In *The Twelfth International Conference on Learning Representations*.

[10] Elmar Juergens, Florian Deissenboeck, Benjamin Hummel, and Stefan Wagner. 2009. Do Code Clones Matter?. In *Proceedings of the 31st International Conference on Software Engineering (ICSE '09)*. IEEE Computer Society, USA, 485–495. doi:10.1109/ICSE.2009.5070547

[11] T. Kamiya, S. Kusumoto, and K. Inoue. 2002. CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code. *IEEE Transactions on Software Engineering* 28, 7 (July 2002), 654–670. doi:10.1109/TSE.2002.1019480

[12] Kisub Kim, Dongsun Kim, Tegawendé F. Bissyandé, Eunjong Choi, Li Li, Jacques Klein, and Yves Le Traon. 2018. FaCoY: A Code-to-Code Search Engine. In *Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 946–957. doi:10.1145/3180155.3180187

[13] 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:2507.15003 [cs] doi:10.48550/arXiv.2507.15003

[14] Junlong Li, Fan Zhou, Shichao Sun, Yikai Zhang, Hai Zhao, and Pengfei Liu. 2024. Dissecting Human and LLM Preferences. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 1790–1811. doi:10.18653/v1/2024.acl-long.99

[15] T.J. McCabe. 1976. A Complexity Measure. *IEEE Transactions on Software Engineering* SE-2, 4 (Dec. 1976), 308–320. doi:10.1109/TSE.1976.233837

[16] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training Language Models to Follow Instructions with Human Feedback. In *Proceedings of the 36th International Conference on Neural Information Processing Systems (NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, 27730–27744.

[17] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. 2023. Do Users Write More Insecure Code with AI Assistants?. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23)*. Association for Computing Machinery, New York, NY, USA, 2785–2799. doi:10.1145/3576915.3623157

[18] Chanchal Kumar Roy and James R. Cordy. 2007. A Survey on Software Clone Detection Research. *Queen's School of computing TR* 541, 115 (2007), 64–68.

[19] Yusuke Sasaki, Tetsuo Yamamoto, Yasuhiro Hayase, and Katsuro Inoue. 2010. Finding File Clones in FreeBSD Ports Collection. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. 102–105. doi:10.1109/MSR.2010.5463293

[20] Mrinank Sharma, Meg Tong, Tomasz Korbak, David Duvenaud, Amanda Askell, Samuel R. Bowman, Esin Durmus, Zac Hatfield-Dodds, Scott R. Johnston, Shauna M. Kravec, Timothy Maxwell, Sam McCandlish, Kamal Ndousse, Oliver Rausch, Nicholas Schiefer, Da Yan, Miranda Zhang, and Ethan Perez. 2023. Towards Understanding Sycophancy in Language Models. In *The Twelfth International Conference on Learning Representations*.

[21] Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Yarin Gal, Nicolas Papernot, and Ross Anderson. 2024. The Curse of Recursion: Training on Generated Data Makes Models Forget. arXiv:2305.17493 [cs] doi:10.48550/arXiv.2305.17493

[22] Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Nicolas Papernot, Ross Anderson, and Yarin Gal. 2024. AI Models Collapse When Trained on Recursively Generated Data. *Nature* 631, 8022 (July 2024), 755–759. doi:10.1038/s41586-024-07566-y

[23] Jerry Wei, Da Huang, Yifeng Lu, Denny Zhou, and Quoc V. Le. 2024. Simple Synthetic Data Reduces Sycophancy in Large Language Models. (Oct. 2024).

[24] Dejiao Zhang, Wasi Ahmad, Ming Tan, Hantian Ding, Ramesh Nallapati, Dan Roth, Xiaofei Ma, and Bing Xiang. 2024. Code Representation Learning At Scale. arXiv:2402.01935 [cs] doi:10.48550/arXiv.2402.01935