Investigating Autonomous Agent Contributions in the Wild: Activity Patterns and Code Change over Time

Razvan Mihai Popescu
r.m.popescu@tudelft.nl
Delft University of Technology
Delft, The Netherlands

David Gros
dgros@ucdavis.edu
University of California, Davis
Davis, USA

Andrei Botocan
a.botocan@student.tudelft.nl
Delft University of Technology
Delft, The Netherlands

Rahul Pandita
rahulpandita@github.com
GitHub
Denver, USA

Prem Devanbu
ptdevanbu@ucdavis.edu
University of California, Davis
Davis, USA

Maliheh Izadi
m.izadi@tudelft.nl
Delft University of Technology
Delft, The Netherlands

Abstract

The rise of large language models for code has reshaped software development. Autonomous coding agents, able to create branches, open pull requests, and perform code reviews, now actively contribute to real-world projects. Their growing role offers a unique and timely opportunity to investigate AI-driven contributions and their effects on code quality, team dynamics, and software maintainability. In this work, we construct a novel dataset of approximately 110,000 open-source pull requests, including associated commits, comments, reviews, issues, and file changes, collectively representing millions of lines of source code. We compare five popular coding agents, including OpenAI Codex, Claude Code, GitHub Copilot, Google Jules, and Devin, examining how their usage differs in various development aspects such as merge frequency, edited file types, and developer interaction signals, including comments and reviews. Furthermore, we emphasize that code authoring and review are only a small part of the larger software engineering process, as the resulting code must also be maintained and updated over time. Hence, we offer several longitudinal estimates of survival and churn rates for agent-generated versus human-authored code. Ultimately, our findings indicate an increasing agent activity in open-source projects, although their contributions are associated with more churn over time compared to human-authored code.

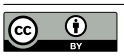
CCS Concepts

• **Software and its engineering → Software post-development issues**.

Keywords

Autonomous Coding Agents, Large Language Models, Mining Software Repositories, Pull Requests, Code Churn, Empirical Software Engineering, Human-AI Collaboration

ACM Reference Format:
Razvan Mihai Popescu, David Gros, Andrei Botocan, Rahul Pandita, Prem Devanbu, and Maliheh Izadi. 2026. Investigating Autonomous Agent Contributions in the Wild: Activity Patterns and Code Change over Time. In *23rd International Conference on Mining Software Repositories (MSR '26), April 13–14, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3793302.3793354



MSR '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2474-9/2026/04
https://doi.org/10.1145/3793302.3793354

1 Introduction

In 2021, early evidence emerged showing the ability of large pre-trained language models (LLMs) to write complete units of code. This was illustrated through the HumanEval dataset [1], which evaluates the task of generating relatively simple Python programs [1]. They reported that GPT-3 could not solve any of the problems, while their proposed Codex model was able to solve 28.8% of the problems. Other contemporaneous work confirmed that LLMs were capable of generating methods in various programming languages [2–4].

The 2022 release of systems such as GitHub Copilot for code completion and ChatGPT for general dialog accelerated the broader use of LLMs for software. However, the assistive nature and limited environmental integration of these early systems have made it challenging to observe and assess their use in the wild, with most research focusing on acceptance rates, which, sadly, fail to capture representative software quality attributes [5–7]. Since then, AI assistants have become increasingly embedded in developer workflows, contributing to higher levels of user satisfaction and productivity [8–13]. Recent works report that over 90% of coding assistants have been released in the past two years [10], while 84% of developers are already using or plan to incorporate AI tools in their development process [14].

With this wave of progress, the software engineering field is transitioning into an AI-native phase, where development is guided by human intent and collaborations with autonomous AI-coding partners [15, 16]. Such emerging agents move beyond passive AI-assistants, being characterized by three core features, including autonomy, expanded scope, and engineering practicality. With improved tooling, these agents can now actively manage and execute development workflows from requirements to implementation [17].

The fast-paced and sheer volume of contributions produced by agentic systems, which already accounts for 10% of public Pull Requests (PR) on GitHub [18], creates the need to examine their activity and understand how these systems might influence factors such as code maintainability, which is traditionally the most resource-intensive phase of the software development life-cycle [19]. Unlike traditional assistants or dialog systems, where mixed developer and

[1]The median is only seven lines-long reference solutions.

AI activity made provenance difficult to trace without access to specialized telemetry or self-admitted usage artifacts [20], agent system's enhanced environment integration enables relatively easier identification of code generated with heavy AI assistance, using their tell-tale signals [16, 18, 21].

In this work, we take advantage of this distinction to mine GitHub PRs written by autonomous coding agents. Previous studies in this area have examined the role of models such as ChatGPT in software development [20, 22, 23], while others have explored the impact of reactive AI assistants on code maintainability through controlled user studies [24] or synthetic benchmarks [25]. However, such approaches remain limited in capturing how these models interact with existing code due to the isolated nature of their evaluation protocols. More recent work investigating the integration of agentic coding in collaborative development remains centered on high-level contribution dynamics, such as differences in PR purpose, acceptance rates, revision effort, or review behavior [16, 18, 21]. More importantly, most of these investigations focus on a *limited* subset of PRs and are often restricted to *highly popular repositories* [16] or a single coding agent [21]. We highlight how, unlike human PRs, most agent PR work is concentrated in very low-star repositories, making a broad view important for understanding these systems. Our study complements current literature by providing a *broader* and *deeper* view of the agentic landscape.

Additionally, we contribute new directions to this understanding by studying the characteristics of agentic activity within collaborative development, with a particular focus on change signals associated with the quality of agent-authored code. This includes a first-of-its-kind longitudinal study of the evolution of agent-authored PR code, focusing on five coding agents, including OpenAI Codex, GitHub Copilot, Claude Code, Google Jules, and Devin. Specifically, our main research questions (RQs) are:

RQ1: What is the difference between agent-authored and human-authored PRs in shaping collaboration and development progress?

RQ2: How do agent-authored PRs influence the trajectory of code maintenance over time compared to human-authored ones?

Our findings indicate that although agent activity is growing in open-source projects, particularly in lower-starred repositories, their contributions are correlated with more extensive code changes over time compared to human-authored code.

Our contributions are as follow:

- We curate a large-scale dataset comprising over 110,000 PRs and related metadata from public GitHub data, capturing the activity and collaboration of humans and coding agents in open-source software development spanning several months.
- We examine the difference in activity dynamics between agent- and human-authored PRs across various dimensions, including merge rates, merge latency, complexity of changes, file composition, commit and review density, and repository characteristics.
- We assess the subsequent evolution and stability of the code contributed by agents and humans using indicators such as *survival rate* and *code churn*.
- We release our code and dataset to support future studies on agentic software development, including human-agent interactions, productivity patterns, and governance of AI contributions. [2]

2 Related Work

Below, we review prior work on AI assistants' evolution into autonomous agents, their evaluation, and their real-world activity and collaborative interactions with human developers.

2.1 From Assistants to Agents

AI assistants are typically defined by their assistive and predictive capabilities, integrated within a developer's workflow, for example, as IDE extensions [16]. These systems can operate as pair programmers designed to support developers across various coding tasks, such as generating code [5], or identifying existing issues or vulnerabilities [26]. Early coding assistants, including Code4Me [5] or IntelliCode Compose [6], powered by generative transformer models such as GPT-C and InCoder, were able to perform short-sequence token predictions and automatically complete entire lines of syntactically correct code in several programming languages. Similarly, GitHub Copilot and Amazon CodeWhisperer could transform natural language comments into lines or entire blocks of source code taking into account code-context [25], while AIBugHunter was able to locate and classify vulnerability types, estimate their severity, and even suggest potential vulnerability fixes to developers [26]. However, all these AI assistants require user confirmation and oversight at each step in their interaction model. Additionally, despite evidence of increased developer productivity and satisfaction [7], in many cases these coding assistants still lack a clear understanding of contextual clues, leading to the introduction of code smells that can cause technical debt in the long term [10, 25].

The improving ability of LLMs, both on code-generation and instruction-following, has fueled a transition from reactive coding assistants to more proactive agents that can reason over broader project contexts. These agents are able to autonomously perform multi-step task planning, take actions, run self-evaluations, and coordinate across multiple tools [27]. A growing number of agentic systems now demonstrate this kind of autonomous coordination across the software development cycle [27]. RepoAgent has been used to proactively generate and maintain comprehensive documentation for entire code repositories [28], while RepairAgent, which leverages a finite state machine to mimic human debugging states, has shown promising results in fixing software bugs [29]. Furthermore, SWE-agent [30] and OpenHands [31] were designed to equip LLMs with specialized interfaces to transform them from reactive predictors into autonomous agents capable of complex, multi-step actions. Accordingly, these models can navigate entire repositories, execute programs, or edit code files [30, 31]; in addition, OpenHands also enables the delegation of tasks to other specialized agents to solve challenging problems.

Additionally, there is a rise of multi-agent systems as well, which mimic the task distribution, specialized roles, and coordination activity of real developer teams. AgentCoder [32] allows agents to take on a different role, including programmers, test designers, and

[2] Code is available at https://github.com/anonym-RD/agents-study

test executors. Similarly, for MaintainCoder [33], which focuses on ensuring high code maintainability, agents are responsible for tasks ranging from requirement analysis and design pattern selection to framework design, evaluation, and code optimization. Lastly, CodeSim [34] addresses program synthesis with its agents focusing on planning, coding, and debugging, introducing a unique simulation-driven approach for evaluating agents' work internally, mirroring how humans visualize and refine algorithms.

These agents not only add a layer of autonomy and persistence over their forerunners, but also possess the ability to adjust their behavior over the observed effects of their own actions while maintaining progress towards the user-provided goal. This is possible through feedback looping, either by internal self-correction, collaboration with humans, or negotiation within multi-agent systems. Nevertheless, this shift from a heavy dependence on developer input to gradually eclipsing it encourages superficial reliance, which over time may lead to undesirable characteristics such as buggy code, or even unmaintainable or security-vulnerable code within the development process.

2.2 Agent Evaluations

To assess AI assistants based on output quality, usability, user satisfaction, or productivity, most prior studies rely on controlled user experiments that evaluate the relative performance of these systems in regards to the developer experience and task goals [35, 36]. Although this evaluation setting is strongly related to the operational nature of these tools, it suffers from low external validity, not entirely capturing the complexity, pace, and nuances of real-world development.

When it comes to agentic systems, static benchmarking represents the primary method to evaluate their capabilities in isolation. Pioneering benchmarks, including HumanEval, MBPP, APPS, and Defects4J [37], are still in the community's attention, despite known issues related to data contamination, erroneous ground truths, or saturation [38, 39]. More recently, a plethora of agentic benchmarks have emerged, targeting different software engineering tasks. SWE-Bench evaluates the ability of agents to solve entire GitHub issues by interacting with repositories [30]. Similarly, SWT-Bench, built on top of SWE-Bench, is specifically designed to evaluate the ability to generate bug reproduction tests [40]. Other benchmarks, such as AgentBench, assess agents' decision-making capabilities in diverse environments, including bash scripting, interacting with real SQL databases, or web browsing [41]. Meanwhile, MaintainBench measures agents' capability to produce maintainable code through requirement evolution cycles [33]. Although these benchmarks provide controlled and reproducible measures of agent performance, they capture only narrow aspects of their development abilities within idealized conditions. This further highlights the need for evaluations that can capture authentic developer-agent interaction and operational project dynamics.

2.3 Agents In The Wild

The growing integration of LLMs into development workflows has motivated efforts to examine their real-world footprint, particularly how these models contribute to and shape activity within repositories. Previous investigations of pull requests, commits, and issues show that developers employ ChatGPT to automate various tasks, including documentation, bug fixing, and code generation [20]. Other studies have examined how LLMs and AI-based tools participate in code reviews, either by assisting developers in providing feedback and alternative solutions [22], or by automatically generating review comments within GitHub workflows [23]. These findings highlight both the growing adoption and the variable effectiveness of AI-generated reviews, emphasizing the need for careful tool design and a better understanding of their broader impact.

Table 1: Search signals used to filter PRs per coding agent.

Coding Agent	Search Signal
OpenAI Codex	head:codex/
GitHub Copilot	head:copilot/
Google Jules	author:google-labs-jules[bot]
Devin	author:devin-ai-integration[bot]
Claude Code	("Co-Authored-By: Claude" OR "Generated with Claude Code")

Following this, other studies have focused on the effects of integrating coding agents into software development workflows. Using the AIDev-Pop dataset, containing 7,122 PRs from GitHub repositories with more than 500 stars and involving agents such as OpenAI Codex, Devin, GitHub Copilot, Cursor, and Claude Code, this work explores productivity patterns [16]. They analyze factors such as resolution and turnaround times, acceptance rates, and structural properties of agent-generated code, including complexity and the nature of the changes. Their results showed that agents can drastically accelerate developer output (e.g., one developer submitted 164 agentic-PRs in three days versus 176 human-PRs over three years), and PRs authored by agents such as Codex are reviewed and merged more quickly than human-authored ones. However, their analysis also reveals that agentic PRs are less likely to be accepted than those of human developers, especially for complex tasks such as feature development or bug fixing.

Complementing this investigation, another work using 567 Claude Code-generated PRs across 157 open-source projects, analyses how these agentic PRs differ from human ones in terms of change size and purpose [21]. Their evaluation shows that agents frequently focus on non-functional improvements such as refactoring, documentation updates, and test additions. Moreover, they observed that most rejections stem from the project context rather than inherent code flows. Crucially, the work quantifies the revision effort, concluding that nearly 55% of agentic PRs are merged without revision.

Nonetheless, these studies offer a limited view of agentic contributions, often focusing on a small set of PRs from a single agent or from highly popular repositories. This creates the need to study activity levels at scale across multiple agents and to investigate the ongoing evolution of agent-generated code.

3 Methodology

This section describes the collection, filtering, and structure of the GitHub dataset, along with the evaluation protocol.

3.1 Data Curation

We target five autonomous coding agents, including Devin (March 2024) [3], Claude Code (February 2025) [4], OpenAI Codex (May 2025) [5], GitHub Copilot (May 2025) [6], and Google Jules (May 2025) [7], due to their growing adoption across both academic and industrial environments, thus forming a representative coverage of current agentic coding paradigms that span both established and more recent agents. Using the GitHub GraphQL *search* API [8], we scrape pull requests associated with these agents, along with other contribution metadata such as changed files, issue references, commits, comments, and reviews from each PR. To ensure coverage of all five agents, we fix an extraction snapshot of three months, covering June through August 2025, according to the announced release dates. Although the time-frame is quote recent, most of these coding agents had already achieved significant usage [18]; looking further back would have included periods when some agents did not yet exist. This window also accounts for earlier agents, such as Devin, which initially lacked PR-related capabilities [9], and Claude Code, first released for research preview in February, before becoming generally available in May. It also includes the newest agent, Google Jules, released in late May, which quickly gained community attention through new features such as critic-augmented generation [10].

To identify which pull requests correspond to which agent, we use different filtering signals according to the tell-tale signature of each agent [16, 18, 21], as illustrated in Table 1. When tackling a given issue, agents such as Codex and Copilot usually operate by creating a new branch prefixed with their name, followed by the issue or requirement title. Since these agents do not have an associated GitHub identity and operate as pseudo-authors, the branch prefix serves as the primary signal to identify related PRs, as shown in prior studies [16, 18]. This approach focuses on the head branch, which contains all proposed changes intended for merging into the base branch. On the other hand, agents such as Google Jules and Devin operate as GitHub bots (registered as applications), meaning their contributions can be filtered using the author field. Lastly, instead of branch prefixing, Claude Code typically appends an authorship watermark to its contributions, which we can track in PR descriptions.

We observed that agents such as Copilot, Claude, and Codex have substantially more PRs during the selected period than the others, with Codex alone reaching approximately 1.1M PRs. As our goal is to characterize the emerging paradigm of PR-based coding agents rather than model adoption trends, we construct a representative sample for each agent covering this time window. According to the GitHub Innovation Graph for Q1 2025 [42], most GitHub developers are based in the United States, followed by the European Union, India, China, and Brazil. To ensure a fair comparison and minimize temporal biases, our scraping traverses the entire three-month period. We adapt the sampling granularity based on the agent's PR volume, using denser sampling for agents with higher PR activity and sparser sampling for those with fewer PRs. In practice, this corresponds to shorter time intervals (e.g., one hour for Codex) and longer intervals (e.g., five hours for Claude and Copilot), which we empirically tuned to achieve balanced coverage throughout the day. To manage days with unusually high activity, we impose a daily upper limit on sampled PRs for each agent, derived from the per-agent total sample (with respect to the least active ones), divided by the number of days. This ensures a reasonable and comparable sample size across all agents, preventing highly active ones from dominating the dataset. To maximize coverage throughout the window, the daily limit is recalculated after each day according to the remaining PRs and the number of days left, ensuring that sampling remains distributed throughout the full period rather than concentrated on a few high-activity days. Due to their lower PR activity and lack of pronounced usage concentration, we included all PRs for Jules and Devin throughout the period.

After collecting all agent-associated PRs and their corresponding repositories (including base and head repositories), we performed manual sanity checks and removed duplicate PRs[11] based on their IDs, accounting for repeated entries introduced by the pagination process. In addition to the absolute counts of PR properties, we also gathered auxiliary metadata for up to the first 100 commits, comments, issues, reviews, and modified files associated with each PR, when available. This limit corresponds to GitHub's maximum page size and avoids rate-limit overhead from nested pagination in the case of unusually large pull requests. Inspection of the distributions also shows that approximately 75% of PRs (across all agents) contain between 2 and 14 entries per metadata type, supporting our threshold choice for capturing complete information in most cases while still accommodating larger PRs.

For comparison, we apply the same scraping procedure to obtain human-created PRs for the same three-month window, using a

Table 2: Search signals for distinguishing agent- and human-generated PRs.

Coding Agent	Search Signal
OpenAI Codex	head:codex/
GitHub Copilot	head:copilot/
Cursor	head:cursor/
Cosine	head:cosine/
Tembo	head:tembo/
OpenHands	head:openhands
Google Jules	author:google-labs-jules[bot]
Devin	author:devin-ai-integration[bot]
Amazon Q	author:amazon-q-developer[bot]
Jetbrains Junie	author:jetbrains-junie[bot]
Junie EAP	author:junie-eap[bot]
Codegen	"About Codegen"
OpenHands	("Co-authored-by: openhands" OR "Automatic fix generated by OpenHands")
Claude Code	("Co-Authored-By: Claude" OR "Generated with Claude Code")

[3] https://cognition.ai/blog/introducing-devin
[4] https://www.anthropic.com/news/claude-3-7-sonnet
[5] https://openai.com/index/introducing-codex/
[6] https://github.blog/news-insights/product-news/github-copilot-meet-the-new-coding-agent/
[7] https://blog.google/innovation-and-ai/models-and-research/google-labs/jules/
[8] https://docs.github.com/en/graphql
[9] https://docs.devin.ai/release-notes/overview
[10] https://jules.google/docs/changelog/2025-08-083/
[11] less than 0.001% of data

Table 3: Dataset summary of pull requests and related contributions. The first three columns indicate the number of PRs created by each author, along with their base and head repositories (hence the large number), while the other four reflect activity associated with those PRs.

PR Author	#PR	#Repository	#Commit	#Comment	#Review	#Issue	#Changed File
OpenAI Codex	20,835	41,669	27,530	3,693	1,957	45	90,822
Claude Code	19,148	38,260	82,755	22,329	12,728	4,052	255,275
GitHub Copilot	18,563	37,125	69,896	26,664	20,665	9,744	158,404
Google Jules	18,468	36,936	41,032	5,700	3,249	2,185	138,610
Devin	14,045	28,090	51,641	27,518	6,901	294	131,454
Human	20,910	41,542	102,037	18,559	21,401	1,973	194,861

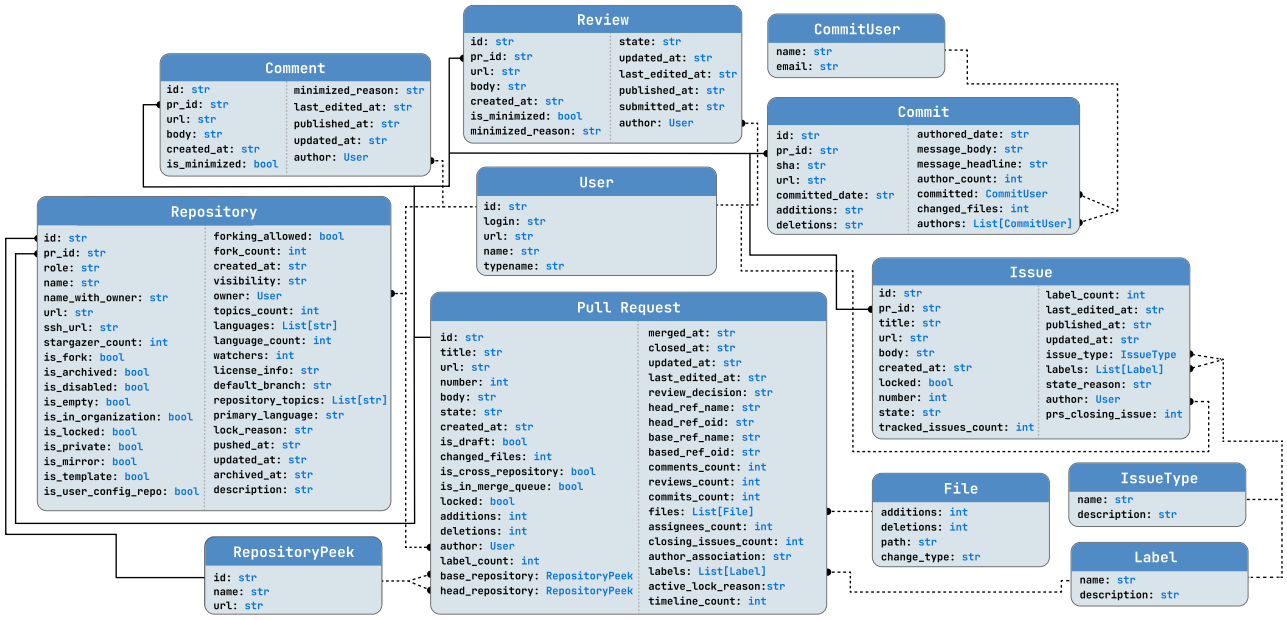


Figure 1: Overview of dataset structure. Solid lines indicate relationships between entities. Dotted lines denote nested objects.

one-hour time interval because of the high PR density. To exclude agent-generated PRs, we filter based on the head branch prefix, authorship, and the presence of agent watermarks in the PR description. Because there is no clear picture of all agents that operate on GitHub, we excluded a set of known agents [18], as well as two additional agents, Amazon Q and JetBrains Junie, and an extra signal for OpenHands, identified through manual GitHub inspection to improve pull request attribution. This list of agents is shown in Table 2, together with the filtering signals used for each of them. To further reduce the risk of misattributing pull requests, we exclude PRs authored by bots or applications (e.g., *dependabot* or *github-actions bot*), which are commonly used for automatic dependency management or other CI/CD tasks.

3.2 Data Structure

Table 3 presents the final number of pull requests per category, together with associated activity information for each PR. The resulting dataset includes 111,969 PRs contributed by both agents and humans. Some variation in activity levels can be seen across agents, with Copilot and Claude PRs appearing more active in the collected samples. Figure 1 illustrates the overall dataset structure, which is described in more detail below.

- `Pull Request`: records the content, state, and activity of a pull request, including author, repository references, timestamps, and total number of commits, reviews, comments, closed issues, labels, and files changed.
- `Repository`: stores a repository's ownership, visibility, status flags, popularity metrics, programming languages, topics, licensing, timestamps, and descriptive information.

- `Commit`: captures a commit's identity, content, timestamps, authoring and committing information, changed files, and associated authors for a given pull request.
- `Review`: lists a pull request review, including its identifier, author, content, state, timestamps, and minimization status.
- `Comment`: represents a pull request comment with its identifier, author, content, timestamps, publication status, and minimization details.
- `Issue`: stores information about an issue linked to a pull request, including its identifier, author, title, description, state, timestamps, type, labels, and other associated PRs.

We also record nested information about the files changed by each pull request. For each file, we store the number of changes in terms of additions and deletions, the type of change, and the absolute path within the repository. Similarly, for users, we capture their public-facing data such as identifier, name, login, profile URL, and user type, which were used for attribution filtering and data validation.

3.3 Evaluation Metrics

3.3.1 Activity Metrics. We start with discussing some of the key high-level metrics, used to address RQ1 [18, 21]. These metrics are computed over the collected pull requests to summarize the activity characteristics of both AI agents and human contributors.

Change Size. This metric measures the magnitude of a pull request in terms of code modifications, using the *additions* and *deletions* attributes. It indicates how substantial the code modifications are within a pull request.

$$\text{Change Size} = \text{Lines Added} + \text{Lines Removed} \tag{1}$$

The value we pull here from the GitHub API reflects the size of the change when it was first opened. The size of the change might evolve through the PR process.

Merge Rate. The merge rate indicates the proportion of submitted pull requests that are successfully merged into the target branch, using the *state* attribute.

$$\text{Merge rate} = \frac{\text{Number of merged PRs}}{\text{Total number of PRs}} \tag{2}$$

As noted in previous investigations, merge rates are strongly influenced by the level of human approval involved in the workflow. For example, GitHub Copilot can open draft PRs on its own, while Codex does so only after human approval [18].

Merge Time. This metric calculates the duration between the opening and merging of a pull request based on the *created_at* and *merged_at* timestamps. It reflects how quickly a contribution is reviewed and accepted into the code base.

$$\text{Merge Time} = t_{\text{merged}} - t_{\text{opened}} \tag{3}$$

Others. We also consider other PR-related measures: the number of comments (*comments_count*) and reviews (*reviews_count*), repository popularity (*stargazer_count*), the fraction of changes that are additions (PR *additions*), and the variety of file types modified (*path*).

3.3.2 Code Change Metrics. We assess code changes using the metrics defined below, applied to commits from both AI agents and human contributors, in line with RQ2.

Survival Rate. The survival rate measures the proportion of code lines that remain unchanged over a period of time. While this metric was previously applied at the repository scale [43], we adapt it to the commit level. Given a seed commit, we extract all line additions in its diff. Next, we identify the nearest subsequent commit within a defined time window. Looking forward from the initial to the target commit, we capture how the newly introduced lines persist in time. We measure survival at the syntactic level: a line is deemed surviving if its content is unchanged in the target commit, and dead otherwise.

$$\text{Survival rate} = \frac{\text{Survived lines}}{\text{Added lines}} \tag{4}$$

A higher survival rate means that the added lines of code were not modified or removed, indicating that their contribution is stable.

Churn Rate. The churn rate indicates the proportion of code lines that were added or modified over a period of time [44]. Similarly, starting from an initial commit, we identify changes from subsequent commits up to a target commit, focusing on modifications in the same files modified by the seed commit. This represents the number of lines that are newly added or syntactically differ from their initial version (Churned LOC), normalized by the total number of changed lines (Total LOC).

$$\text{Churn rate} = \frac{\text{Churned LOC}}{\text{Total LOC}} \tag{5}$$

A higher churn rate reflects code has been more reworked, which may raise concerns about maintainability and stability.

Deletion Rate. The deletion rate captures the proportion of code lines removed over a period of time [44]. While this is similar to churn, it focuses on the number of lines deleted from the initial commit (Deleted LOC), normalized by the total number of changed lines (Total LOC).

$$\text{Deletion rate} = \frac{\text{Deleted LOC}}{\text{Total LOC}} \tag{6}$$

A higher deletion rate reflects possible refactoring or restructuring.

4 Results and Discussion

In the following, we present our results, first by analyzing the activity patterns of the coding agents and then exploring how their contributions affect maintainability signals over time.

4.1 RQ1: Agents' Contributions

4.1.1 Pull Request and Repositories Characteristics. We start by describing the types of repositories that appear in our sample.

GitHub Repository Stars: The number of stars of a repository is a signal of its popularity, community engagement, and maturity. We explore some of the details of this in Table 4 for the BASE (i.e., the merged into) repositories. We observe that 40.5% of the human PRs in our sample are from repositories with 0 stars. Agent PRs are more likely to be in 0-star repositories. In the case of Codex and Jules, about 75% of PRs in our sample have 0 stars. While PRs

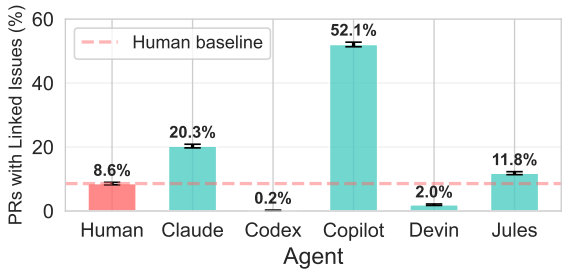


Figure 2: Fraction of PRs linked to issues. Wilson 95% CIs shown.

Table 4: Distribution of pull requests across repository stars, with most PRs originating from repositories with 0 stars. Human PRs tend to appear in higher-starred repositories.

PR Author	PR's Repository Star Count				
	%=0	%≥10	%≥100	Median	Med(if>0)
Claude	51.7%	19.7%	10.4%	0	5
Codex	75.3%	5.2%	2.4%	0	1
Copilot	59.6%	15.2%	9.1%	0	3
Devin	64.1%	23.0%	14.3%	0	39
Jules	75.7%	4.0%	1.4%	0	1
Human	40.5%	36.8%	25.0%	1	38

are traditionally used in popular repositories for collaborating with other developers, the rise of agents is reshaping the PR usage to also be in developers' personal or small repositories.

While it is more rare that agent PRs occur in very popular 100+ star repositories, it does occur in some cases, approaching about 15% of an agent's PRs. The overall median stars are close to 0. Although interestingly, when taking the median excluding 0-star repositories, the Devin median more closely matches that of the human PRs. We might speculate that because Devin is a slightly older and relatively more expensive and specialized tool, there is a larger proportion of cases where it is applied to higher-starred repositories. In contrast, PR agents like Copilot and Codex are more directly in front of every developer with less separate cost due to their GitHub and ChatGPT integration, and thus are more likely to be used in small projects.

Issues: Different agents encourage different patterns. The fraction of linked issues is shown in Figure 2, with 95% Wilson [45] confidence intervals (CIs) for statistical significance.

Linked issues are relatively uncommon in GitHub PRs, appearing in only about 8% of human submissions. In contrast, agents such as Copilot include linked issues in roughly half of their PRs, as assigning an issue is a central aspect of the agent's UX, though it can also be invoked through a chat interface. Meanwhile, Codex and Devin rarely have assigned issues.

4.1.2 Pull Request Changes. Next, we examine the specific changes introduced in these PRs.

Change Size: Figure 3 shows summary statistics for the number of lines added or deleted in sampled PRs connected to each agent. In

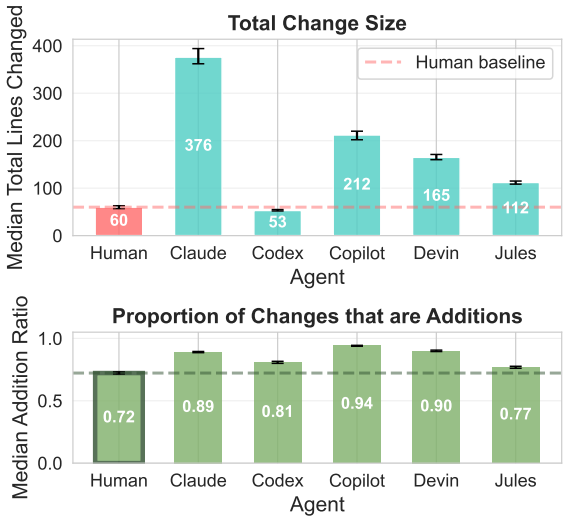


Figure 3: The plot at the top shows the median PR change size for each agent, while the bottom one shows the median fraction of changes that are additions. Bootstrapped 95% CIs shown.

standard diff format, a modified line is recorded as both a deletion and an addition, so it counts twice toward the total lines changed.

We observe that in our sample, agent edits have progressed beyond just simple edits. The median human-opened PR was approximately 60 lines. Generally agents were used in PRs with more lines changed (Claude median 376 lines), with the exception of Codex, which was used in smaller changes. This is one of the indicators of how Codex is often used differently than some of the other agents.

Our large sample size lends high statistical power, making differences significant[12]. However, there is fairly high variance, so the actual effect size can be moderate to fairly negligible. A Cliff's Delta (δ) [46] between Claude and human change sizes is about $\delta = 0.37$ and about $\delta = -0.03$ between human and Codex. For all agents, we observe higher rates of adding code compared to deleting code relative to our human sample. Due to the between-PR variance, effect sizes are fairly small (e.g., human-Copilot $\delta = 0.26$).

Overall, these change-size statistics suggest that agent-generated PRs tend to involve larger, addition-heavy changes, although this is not every PR. The distribution has a long tail of change lines size, so we see PRs for all authors of many sizes.

File Type: In Figure 4, we explore what file types are appearing in the PRs. Our scraping limits to the first 100 files in the PR, and reflects touched files from comparing the latest PR HEAD to the BASE when the PR was created.

In line with the general popularity numbers [47], we observe that JavaScript and TypeScript (TS/JS) are the most likely group of file types to appear in observed PRs. [13] They occur in approximately

[12] A pairwise Mann-Whitney U test of change size differences estimates all $p \ll 0.0001$

[13] We opt to combine JS/TS. While these are not equivalent languages, they both reflect closely related ecosystems and usage patterns.

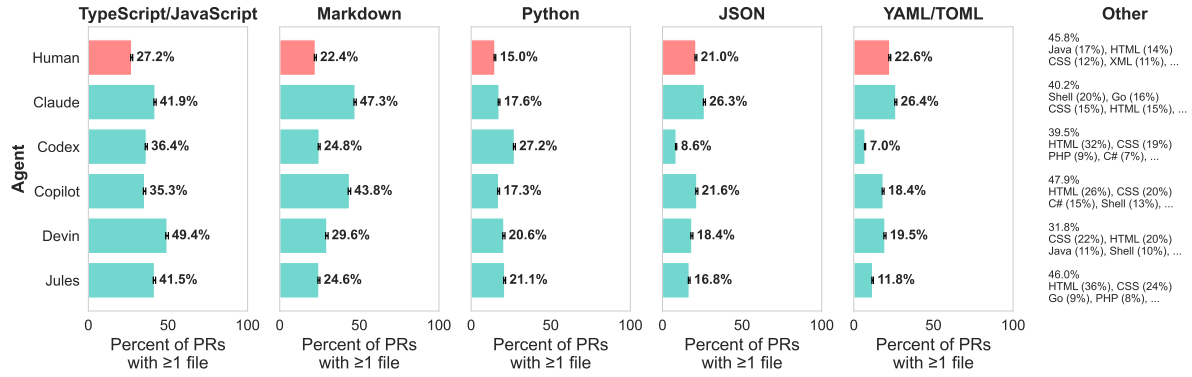


Figure 4: Percentage of pull requests containing the corresponding file type. The five most common types are shown (related types grouped, e.g., TS/JS, YAML/TOML); less common types are grouped as "Other" with right-margin annotations listing the next four most frequent types. Wilson 95% confidence intervals are shown.

a quarter of human PRs, but over a third of agent PRs. In particular, Devin has a high fraction of TS/JS PRs.

Python is the second most common programming language representing 15% of human PRs. Agent PRs are more likely to have Python code than human PRs, particularly Codex showing a high percent of Python PRs (27%).

We also observed that agents frequently make changes to non-code files. Markdown file edits appear in close to half of Claude PRs, with all agents having a higher occurrence of Markdown than in human PRs. This can be a reflection of the propensities of agents to add documentation, but also might reflect that the agents are more likely to be used for "0 → 1" type edits, where creating new repositories or independent features from scratch. While these are the most common file types to be included, there is still a long tail of other file types included in PRs.

4.1.3 Pull Request Merges. Next we explore what happens to these PRs after being opened.

Merge Rate: In Figure 5, we compare merge rates across agents, shown both for the full dataset and stratified by repository popularity (right), grouped into three star-count tertiles. Each PR can be in one of three states: open, merged, or closed. We show the percentage of all PRs that are merged out of all PRs (as PRs that are left open can represent cases where agents did not resolve an issue, but caused developer effort).

Across all the data, we observe that agents such as Claude and Codex are merged at a higher rate than human PRs in our sample, while Copilot and Devin are merged at a lower rate (we note significantly separated CIs in Figure 5). When looking at the star count, we observe different patterns depending on the PR author. Human, Copilot, and Devin PRs follow a slight "inverted-u" pattern; the 1st tertile and 3rd tertile have lower merge rates than the 2nd. The merge rate for Codex is roughly similar across star counts. Jules merge rate is fairly flat but falls off at higher star counts (notably, Jules is also the agent where our sampling finds the fewest PRs from high-starred repositories).

Merge Time: In Figure 6 we show the distribution of times between PR opening and merge events. A typical PR in our sample is merged fairly quickly. The median human PR is merged in 0.4 hours, and a large fraction (43.3%) of PRs are merged in under 10 minutes. These ranges are somewhat typical for other agents except for Codex and Jules. The median Codex PR in our sample was merged in 0.5 minutes.

If we consider merge times grouped into three categories by repository star count (right side of Figure 5), we observe an increasing trend. In the case of humans, the median PR is merged in only a few minutes. This increases to a median of about 10 hours in higher starred repositories. Although the merge time for some agents' PRs (such as Copilot and Devin) also shifts up with higher starred repositories, the p75 of merge times for Codex remains under 1 hour even for higher starred repositories.

Consistent with the change size findings, this offers an additional perspective on how Codex (and to some extent Jules) is being used differently than other agents. We speculate that the faster merging might arise from agent PRs performing much simpler (or rote) work, or that the agent code is reviewed by humans, off-line, before PR submission, and using PRs only as the AI system's provided mechanism for integrating code into the codebase. The distribution of Copilot PR merge time in high-starred repositories is surprisingly similar to human PRs (thus, they might be receiving similar amounts of review effort), but there is an approximate 27 points gap in merge rates.

High-level Interaction Metrics In Figure 7, we look at some of the measures of how developers interact with PRs. Some agents (such as Devin) will always comment on their own PRs, while others will not. Thus, we focus on comments labeled by GitHub as being from a user rather than a bot.

Comments are fairly rare. Only 21% percent of human PRs in our sample receive comments. We observe that agent PRs typically receive fewer comments (with codex PRs rarely commented on). We also note that human PRs receive fewer reviews than agent PRs (although Co pilot's PRs get marginally more reviews). In a traditional framing of PRs, this would perhaps hint that the PRs and associated changes are simpler and do not require much review. However, agent systems can alter this paradigm. For example, Codex and Jules review might have been more likely to occur off-platform in a

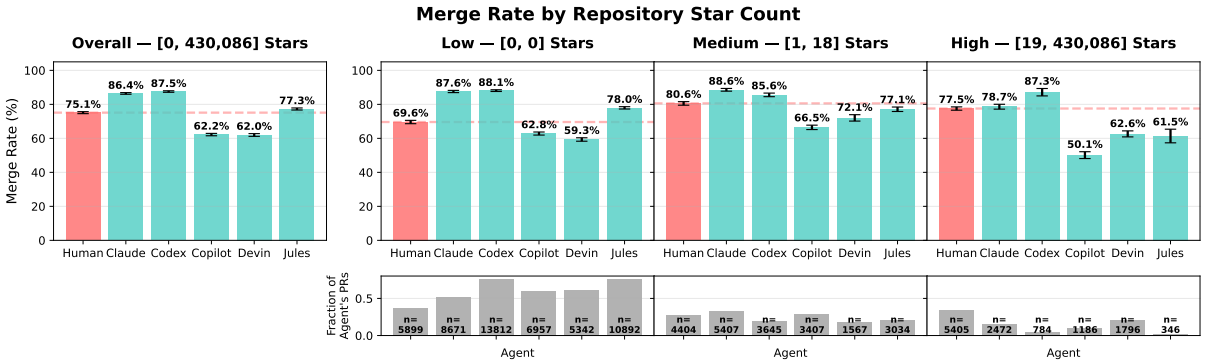


Figure 5: Percentage of pull requests merged per agent, grouped by repository star count (Wilson 95% CIs shown). Lower panel shows each agent's PR fraction within star categories (p33 and p66 of human data). Some agents show an "inverted-u" trend across star counts.

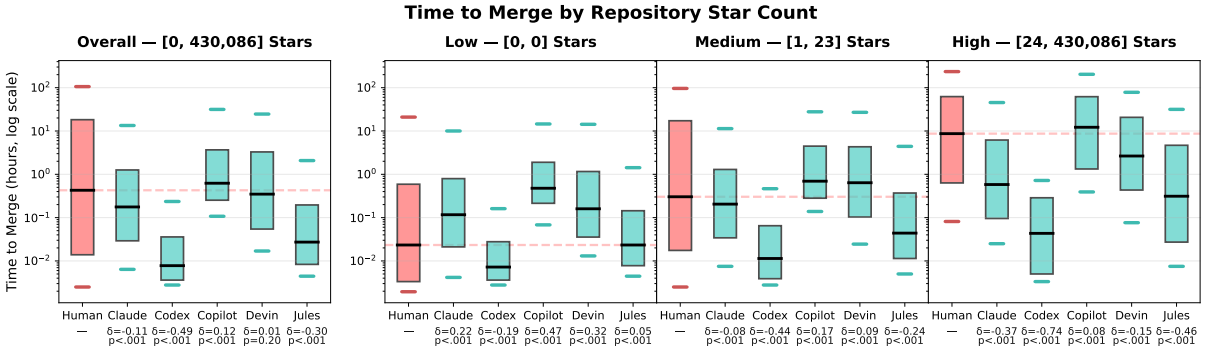


Figure 6: Distribution of merge times across agents (log scale) as boxplots with p10–p90 whiskers. Cliff's δ and Mann-Whitney U test p-values comparing to humans are labeled. Merge times generally increase with repository star count; Codex and Jules PRs merge faster, while shorter times may reflect prior human interaction. Star boundaries are based on human PR tertiles of merged PRs.

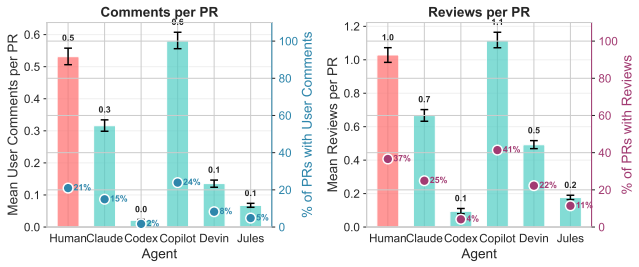


Figure 7: Developer interactions overview with pull requests, showing average counts of user comments (i.e., non-bot comments), reviews (main bar, left axis), and the proportion of PRs with at least 1 comment or review (dots, right axis). Bootstrapped 95% CIs shown.

chat interface, while Copilot is more closely tied with reviews as the mechanism to invoke the system for edits. Additionally, systems such as Copilot include an UI for using the system itself as a reviewer.

4.2 RQ2: Code Change over Time

We examine code evolution at three post-commit intervals: three days, one week, and three weeks. We restrict our analysis to pull requests merged into the code base, as our objective is to understand how successfully-integrated code changes over time. These time intervals were chosen to capture short-range changes following code integration, without extending the analysis into substantially longer horizons (e.g., several months or years), which would require additional longitudinal data. While code may continue to change over longer time spans, our study focuses on this early post-merge window as a snapshot of current agent-driven development practices.

Furthermore, accurately identifying the author of commits can be challenging for some agents. For example, Claude Code allows its watermarks to be disabled, while OpenAI Codex often records commits under developers' accounts [16]. To minimize uncertainty and ensure comparability, we consider the commit reflecting the

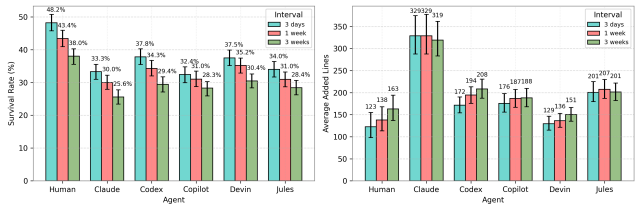


Figure 8: Left plot denotes the fraction of commits with complete line survival across agents measured at 3-day, 1-week, and 3-week intervals. The right plot shows the average net additions per agent across the same time intervals. Bootstrapped 95% CIs shown.

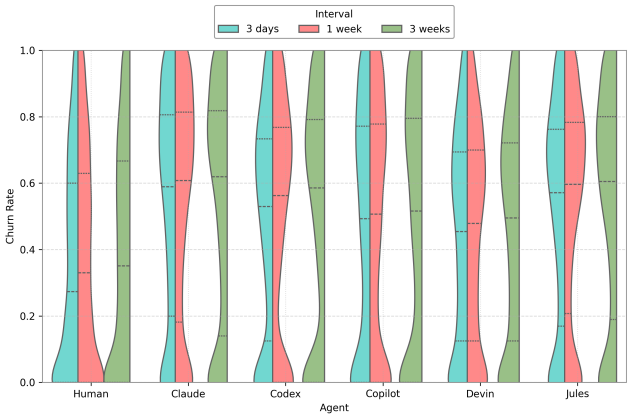


Figure 9: Per-agent distribution of commit churn rates measured over 3-day, 1-week, and 3-week intervals.

agent's initial implementation of the assigned issue, typically the first commit in the PR. Since we know the PR authorship to be agent-generated from our filtering of observable signals, this commit represents the agent's first contribution to the PR prior to review, reducing the risk of confounded authorship. However, we note that for GitHub Copilot, the first commit in a PR is commonly a placeholder used to initialize an issue plan without introducing any file changes. To address this, we instead select the second commit from each Copilot PR, which reflects the actual implementation of the requirement. On top of that, we exclude any commits with no file changes across all agents, if any.

Finally, we sample 1560 commits for each agent and for humans. To reduce bias from repositories with zero stars, we perform stratified random sampling based on repository popularity. Repositories with zero stars form a distinct category, while those with at least one star are sampled according to quantile-based star distributions.

Code Survival. In Figure 8, the left plot shows the percentage of commits where all their initial additions have survived, across all agents. For all three intervals, we can see that the fraction of commits where all lines survived is consistently higher for humans than for any agent, with nearly half of their additions surviving to three days. Mann-Whitney U tests show these differences are

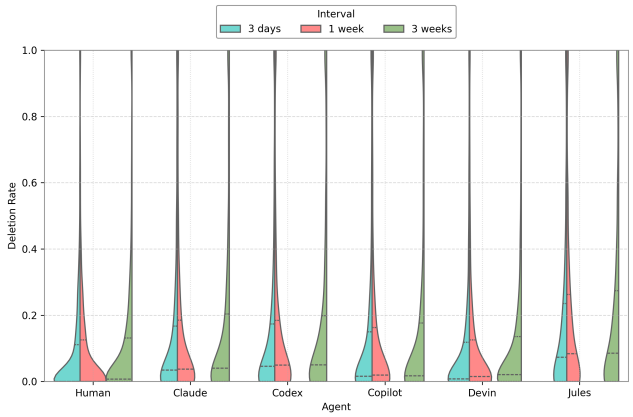


Figure 10: Per-agent distribution of commit deletion rates measured over 3-day, 1-week, and 3-week intervals.

statistically significant ($p < 0.001$), and Cliff's δ values (-0.05 to -0.14) suggest the effect sizes are small but consistently negative, indicating that human-generated changes might persist slightly longer. In contrast, agent-generated commits are more likely to have their lines reworked, suggesting a lower stability of their contributions. Although the overall decline in survival over time for both humans and agents reflects the natural attrition of code, the discrepancy between them could be due to how agents are used in practice. The drop in survival for agent-generated commits may stem from their changes being more localized and repetitive. It is worth noting that commits from agents tend to produce a larger net code increase over the observation interval, as shown in the right plot of Figure 8. This may result from agents generating larger code snippets per commit. Alternatively, human commits likely involve more deliberate and distributed design decisions, which could explain their higher commit survival.

Code Churn. In Figure 9, we illustrate the distribution of commit churn rates for each agent across the three observation intervals. For humans, the distribution is concentrated mainly between 0 and 0.4, indicating that most commits introduce relatively few changes to the total modified files. This suggests that overall human contributions are more stable than agent ones, with less volatility in code modifications. On the other hand, agent commits display a higher median churn compared to humans across all time periods, indicating that their changes are more frequent and larger in proportion to the total changed lines. This tendency is more evident in Claude, which shows pronounced concentration of churn values between 0.8 and 1. This behavior could be associated with the tendency of Claude Code to modify larger portions of the code base in each commit, as we have seen in Section 4.1.2, which could potentially increase the occurrence of later revisions. This high number of changes could stem from formatting, documentation (such as more markdown files as seen in Figure 4), or stylistic adjustments, which are common in automatically generated code.

Furthermore, the distributions for agents themselves remain largely consistent across the three time intervals, with only a slight upward shift in median churn. This suggests that most rework

occurs within the first few days after a commit, and that code tends to stabilize relatively quickly after. Compared to the subsequent decline in commit survival over longer intervals, the stable churn indicates that individual commits generally involve incremental changes, while the later survival drop reflects the cumulative effect of other commits on previously added lines rather than large-scale rework in any single commit.

Additionally, Figure 10 shows the distribution of line deletion rates per commit for each agent. Although deletion rates are low, agent commits consistently exhibit slightly higher deletion ratios than human ones, suggesting that auto-generated code is more frequently replaced or refined. Similar to the survival rate, statistical tests show a significant difference between human and agent commits in terms of churn and deletion rates across all intervals, with effect sizes remaining small ($\delta \approx 0.04$–0.29). Taken together, these findings indicate that while human contributions might be more stable, agent-generated commits are more susceptible to early changes and deletions. Note that this represents only one perspective on the evolution of agent-generated code. Future efforts should examine this phenomenon in greater depth; for example, by analyzing the drivers of code churn (bug fixes, feature additions, requirements changes, or flawed abstractions), and by more clearly distinguishing the types of changes made in real-world PRs, such as code, comments, and documentation. Such analyses would further deepen our understanding.

5 Future Work

The rapid evolution of agentic development offers numerous directions for future research. Beyond analyzing agentic activity in practice, our dataset provides a foundation for benchmarking the quality, stability, and collaboration patterns of agent-generated code. Researchers can use it to fine-tune or evaluate models on tasks such as code generation and repair, using real-world outcomes, such as post-merge churn or acceptance rates, as objective signals. The dataset can also be extended to span longer periods, include new agents, and integrate richer metadata to enable studies of behavioral changes over time and across diverse projects. Future work might also incorporate CI/CD metrics (e.g., build and test results), PR timelines (e.g., review iterations), and contributor information. The long-term effects of agent-generated code could also be studied through analysis of technical debt accumulation and its relationship to agent versions, project characteristics, and review practices.

6 Threats to Validity

System Version Changes Opaque. This is a rapidly evolving area. Even as we scraped our data, these agent systems were actively being tweaked and modified. Our findings partially accommodate this by focusing on aggregated behavioral patterns across all pull requests, capturing the emergent, longitudinal behavior of agentic development rather than individual version-specific effects.

Agent Contribution Transparency. Agents such as OpenAI Codex and Claude Code can make untraceable contributions via their CLI interface, and any authorship watermarks that might signal these contributions can be disabled. Particularly for Codex, activity is usually recorded under human accounts, making it difficult to fully track more granular contributions. To account for this, our analysis focuses only on pull requests that can be reliably attributed to agents and, therefore, reflects only the activity that is observable rather than the absolute range of agent contributions. Furthermore, for the code change analysis, we specifically select the initial commits from merged PRs created by agents, to minimize the risk of confounded authorship, while recognizing that precise attribution remains an open challenge, especially as agent-based workflows on GitHub are still emerging.

Human Attribution Uncertainty. Since the full set of active coding agents on GitHub is unknown, filtering for purely human-authored data may still include undetected agentic contributions. To reduce this threat, we identify PRs using signals consistent with prior work, supplemented with additional agents identified through manual inspection to improve coverage of the agentic landscape at the time of the study. We also exclude any PRs created by bots or applications that are not real users, and we apply the same commit authorship strategy used for agents. Together, these measures collectively remove most automated activities.

Repository Popularity Imbalance. The collected PRs show a skewed distribution of repository popularity, particularly across agents, with most PRs originating from low-star repositories. This pattern reflects the distribution seen in the wild and aligns with prior work [18]. In the activity analysis, we report results by repository popularity levels to show how activity patterns vary in practice. For the code change analysis, we mitigate potential bias from repository popularity by using stratified sampling across different popularity levels.

Data Collection Coverage. For agents with a large number of PRs, the GitHub API may return contributions concentrated within specific time frames, which may lead to the dataset underrepresenting the global distribution of agentic contributions. To mitigate this, we uniformly sample PRs across days and time intervals within each day, proportionally to the total number of PRs for each agent.

7 Conclusion

AI coding agents, although a recent phenomenon, are evolving rapidly and already shaping software development. Public PRs from these agents offer a valuable lens on this shift, despite UX and usage differences that make comparability between agents and humans challenging. Our longitudinal study of PRs suggests that agent-generated code is significantly less likely to be retained and more prone to churn. We also observe a dramatic rise in agent generated PRs targeting zero-star repositories. Although such PRs behave differently from the PRs in more popular repositories, they cannot be ignored, as they currently represent the majority of agent activity. Whether these agents will see wider adoption in higher-starred repositories remains uncertain. The reasons for this pattern require further investigation, but the findings may have important implications for assessing the overall costs and benefits of automated agents. We hope that our study adds to the initial understanding of this important emerging trend.

References

[1] Mark Chen et al. 2021. Evaluating large language models trained on code. (2021). https://arxiv.org/abs/2107.03374 arXiv: 2107.03374 [cs.LG].

[2] Jacob Austin et al. 2021. Program synthesis with large language models. (2021). https://arxiv.org/abs/2108.07732 arXiv: 2108.07732 [cs.PL].

[3] Dan Hendrycks et al. 2021. Measuring coding challenge competence with apps. (2021). https://arxiv.org/abs/2105.09938 arXiv: 2105.09938 [cs.SE].

[4] Federico Cassano et al. 2023. Multipl-e: a scalable and polyglot approach to benchmarking neural code generation. *IEEE Trans. Softw. Eng.*, 49, 7, (July 2023), 3675–3691. doi:10.1109/TSE.2023.3267446.

[5] Maliheh Izadi, Jonathan Katzy, Tim Van Dam, Marc Otten, Razvan Mihai Popescu, and Arie Van Deursen. 2024. Language models for code completion: a practical evaluation. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (ICSE '24) Article 79. Association for Computing Machinery, Lisbon, Portugal, 13 pages. ISBN: 9798400702174. doi:10.1145/3597503.3639138.

[6] Maliheh Izadi, Jonathan Katzy, Tim Van Dam, Marc Otten, Razvan Mihai Popescu, and Arie Van Deursen. 2024. Language models for code completion: a practical evaluation. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (ICSE '24) Article 79. Association for Computing Machinery, Lisbon, Portugal, 13 pages. ISBN: 9798400702174. doi:10.1145/3597503.3639138.

[7] Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2022. Productivity assessment of neural code completion. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming* (MAPS 2022). Association for Computing Machinery, San Diego, CA, USA, 21–29. ISBN: 9781450392730. doi:10.1145/3520312.3534864.

[8] Ebtesam Al Haque, Chris Brown, Thomas D. LaToza, and Brittany Johnson. 2025. The evolution of information seeking in software development: understanding the role and impact of ai assistants. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering* (FSE Companion '25). Association for Computing Machinery, Clarion Hotel Trondheim, Trondheim, Norway, 1494–1502. ISBN: 9798400712760. doi:10.1145/3696630.3731665.

[9] Wenhan Lyu, Yimeng Wang, Yifan Sun, and Yixuan Zhang. 2025. Will your next pair programming partner be human? an empirical evaluation of generative ai as a collaborative teammate in a semester-long classroom setting. In *Proceedings of the Twelfth ACM Conference on Learning @ Scale* (L@S '25). Association for Computing Machinery, Palermo, Italy, 83–94. ISBN: 9798400712913. doi:10.1145/3698205.3729544.

[10] Yunbo Lyu, Zhou Yang, Jieke Shi, Jianming Chang, Yue Liu, and David Lo. 2025. "my productivity is boosted, but ..." demystifying users' perception on ai coding assistants. (2025). https://arxiv.org/abs/2508.12285 arXiv: 2508.12285 [cs.SE].

[11] Gal Bakal, Ali Dasdan, Yaniv Katz, Michael Kaufman, and Guy Levin. 2025. Experience with github copilot for developer productivity at zoominfo. (2025). https://arxiv.org/abs/2501.13282 arXiv: 2501.13282 [cs.SE].

[12] Ya Gao and GitHub Customer Research. 2024. Research: quantifying github copilot's impact in the enterprise with accenture. (May 13, 2024). https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-in-the-enterprise-with-accenture/.

[13] JetBrains. 2024. 2024 developer ecosystem report. https://www.jetbrains.com/lp/devecosystem-2024/.

[14] Stack Overflow. 2025. 2025 stack overflow developer survey. https://survey.stackoverflow.co/2025/.

[15] Ahmed E. Hassan, Gustavo A. Oliva, Dayi Lin, Boyuan Chen, Zhen Ming, and Jiang. 2024. Towards ai-native software engineering (se 3.0): a vision and a challenge roadmap. (2024). https://arxiv.org/abs/2410.06107 arXiv: 2410.06107 [cs.SE].

[16] 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. (2025). https://arxiv.org/abs/2507.15003 arXiv: 2507.15003 [cs.SE].

[17] Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, and Ge Li. 2025. A survey on code generation with llm-based agents. (2025). https://arxiv.org/abs/2508.00083 arXiv: 2508.00083 [cs.SE].

[18] Mark Niklas Müller Christian Mürtz. 2025. Agents in the wild - dashboard. https://insights.logicstar.ai. Interactive web dashboard. Code available at https://github.com/logic-star-ai/insights. (2025). doi:10.5281/zenodo.15846865.

[19] Bo Lin, Shangwen Wang, Zhongxin Liu, Yepang Liu, Xin Xia, and Xiaoguang Mao. 2023. Cct5: a code-change-oriented pre-trained model. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (ESEC/FSE 2023). Association for Computing Machinery, San Francisco, CA, USA, 1509–1521. ISBN: 9798400703270. doi:10.1145/3611643.3616339.

[20] Rosalia Tufano, Antonio Mastropaolo, Federica Pepe, Ozren Dabic, Massimiliano Di Penta, and Gabriele Bavota. 2024. Unveiling chatgpt's usage in open source projects: a mining-based study. In *Proceedings of the 21st International Conference on Mining Software Repositories* (MSR '24). Association for Computing Machinery, Lisbon, Portugal, 571–583. ISBN: 9798400705878. doi:10.1145/3643991.3644918.

[21] Miku Watanabe, Hao Li, Yutaro Kashiwa, Brittany Reid, Hajimu Iida, and Ahmed E. Hassan. 2025. On the use of agentic coding: an empirical study of pull requests on github. (2025). https://arxiv.org/abs/2509.14745 arXiv: 2509.14745 [cs.SE].

[22] Miku Watanabe, Yutaro Kashiwa, Bin Lin, Toshiki Hirao, Ken'Ichi Yamaguchi, and Hajimu Iida. 2024. On the use of chatgpt for code review: do developers like reviews by chatgpt? In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering* (EASE '24). Association for Computing Machinery, Salerno, Italy, 375–380. ISBN: 9798400717017. doi:10.1145/3661167.3661183.

[23] Kexin Sun, Hongyu Kuang, Sebastian Baltes, Xin Zhou, He Zhang, Xiaoxing Ma, Guoping Rong, Dong Shao, and Christoph Treude. 2025. Does ai code review lead to code changes? a case study of github actions. (2025). https://arxiv.org/abs/2508.18771 arXiv: 2508.18771 [cs.SE].

[24] Markus Borg, Dave Hewett, Nadim Hagatulah, Noric Couderc, Emma Söderberg, Donald Graham, Uttam Kini, and Dave Farley. 2025. Echoes of ai: investigating the downstream effects of ai assistants on software maintainability. (2025). https://arxiv.org/abs/2507.00788 arXiv: 2507.00788 [cs.SE].

[25] Burak Yetiştiren, Işık Özsoy, Miray Ayerdem, and Eray Tüzün. 2023. Evaluating the code quality of ai-assisted code generation tools: an empirical study on github copilot, amazon codewhisperer, and chatgpt. (2023). https://arxiv.org/abs/2304.10778 arXiv: 2304.10778 [cs.SE].

[26] Michael Fu, Chakkrit Tantithamthavorn, Trung Le, Yuki Kume, Van Nguyen, Dinh Phung, and John Grundy. 2023. Aibughunter: a practical tool for predicting, classifying and repairing software vulnerabilities. *Empirical Softw. Engg.*, 29, 1, (Nov. 2023), 33 pages. doi:10.1007/s10664-023-10346-3.

[27] Junwei Liu, Kaixin Wang, Yixuan Chen, Xin Peng, Zhenpeng Chen, Lingming Zhang, and Yiling Lou. 2024. Large language model-based agents for software engineering: a survey. (2024). https://arxiv.org/abs/2409.02977 arXiv: 2409.02977 [cs.SE].

[28] Qinyu Luo et al. 2024. Repoagent: an llm-powered open-source framework for repository-level code documentation generation. (2024). https://arxiv.org/abs/2402.16667 arXiv: 2402.16667 [cs.CL].

[29] Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. 2025. Repairagent: an autonomous, llm-based agent for program repair. In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering* (ICSE '25). IEEE Press, Ottawa, Ontario, Canada, 2188–2200. ISBN: 9798331505691. doi:10.1109/ICSE55347.2025.00157.

[30] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2025. Swe-agent: agent-computer interfaces enable automated software engineering. In *Proceedings of the 38th International Conference on Neural Information Processing Systems* (NIPS '24) Article 1601. Curran Associates Inc., Vancouver, BC, Canada, 125 pages. ISBN: 9798331314385.

[31] Xingyao Wang et al. 2025. Openhands: an open platform for ai software developers as generalist agents. (2025). https://arxiv.org/abs/2407.16741 arXiv: 2407.16741 [cs.SE].

[32] Dong Huang, Jie M. Zhang, Michael Luck, Qingwen Bu, Yuhao Qing, and Heming Cui. 2024. Agentcoder: multi-agent-based code generation with iterative testing and optimisation. (2024). https://arxiv.org/abs/2312.13010 arXiv: 2312.13010 [cs.CL].

[33] Zhengren Wang, Rui Ling, Chufan Wang, Yongan Yu, Sizhe Wang, Zhiyu Li, Feiyu Xiong, and Wentao Zhang. 2025. Maintaincoder: maintainable code generation under dynamic requirements. (2025). https://arxiv.org/abs/2503.24260 arXiv: 2503.24260 [cs.SE].

[34] Md. Ashraful Islam, Mohammed Eunus Ali, and Md Rizwan Parvez. 2025. CodeSim: multi-agent code generation and problem solving through simulation-driven planning and debugging. In *Findings of the Association for Computational Linguistics: NAACL 2025*. Luis Chiruzzo, Alan Ritter, and Lu Wang, (Eds.) Association for Computational Linguistics, Albuquerque, New Mexico, (Apr. 2025), 5113–5139. ISBN: 979-8-89176-195-7. doi:10.18653/v1/2025.findings-naacl.285.

[35] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded copilot: how programmers interact with code-generating models. *Proc. ACM Program. Lang.*, 7, OOPSLA1, Article 78, (Apr. 2023), 27 pages. doi:10.1145/3586030.

[36] Joel Becker, Nate Rush, Elizabeth Barnes, and David Rein. 2025. Measuring the impact of early-2025 ai on experienced open-source developer productivity. (2025). https://arxiv.org/abs/2507.09089 arXiv: 2507.09089 [cs.AI].

[37] René Just, Darioush Jalali, and Michael D. Ernst. 2014. Defects4j: a database of existing faults to enable controlled testing studies for java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis* (ISSTA 2014). Association for Computing Machinery, San Jose, CA, USA, 437–440. ISBN: 9781450326452. doi:10.1145/2610384.2628055.

[38] Martin Riddell, Ansong Ni, and Arman Cohan. 2024. Quantifying contamination in evaluating code generation capabilities of language models. (2024). https://arxiv.org/abs/2403.04811 arXiv: 2403.04811 [cs.SE].

[39] Shuo Yang, Wei-Lin Chiang, Lianmin Zheng, Joseph E. Gonzalez, and Ion Stoica. 2023. Rethinking benchmark and contamination for language models with rephrased samples. (2023). https://arxiv.org/abs/2311.04850 arXiv: 2311.04850 [cs.CL].

[40] Niels Mündler, Mark Niklas Müller, Jingxuan He, and Martin Vechev. 2025. Swt-bench: testing and validating real-world bug-fixes with code agents. (2025). https://arxiv.org/abs/2406.12952 arXiv: 2406.12952 [cs.SE].

[41] Xiao Liu et al. 2025. Agentbench: evaluating llms as agents. (2025). https://arxiv.org/abs/2308.03688 arXiv: 2308.03688 [cs.AI].

[42] GitHub. 2025. Developers. https://innovationgraph.github.com/global-metrics/developers.

[43] Adem Ait, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2022. An empirical study on the survival rate of github projects. In *Proceedings of the 19th International Conference on Mining Software Repositories* (MSR '22). Association for Computing Machinery, Pittsburgh, Pennsylvania, 365–375. ISBN: 9781450393034. doi:10.1145/3524842.3527941.

[44] N. Nagappan and T. Ball. 2005. Use of relative code churn measures to predict system defect density. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.* 284–292. doi:10.1109/ICSE.2005.1553571.

[45] Edwin Bidwell Wilson. 1927. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22, 209–212. https://api.semanticscholar.org/CorpusID:121572396.

[46] Norman Cliff. 1993. Dominance statistics: ordinal analyses to answer ordinal questions. *Psychological Bulletin*, 114, 494–509. https://api.semanticscholar.org/CorpusID:120113824.

[47] GitHub Staff. 2024. Octoverse: ai leads python to top language as the number of global developers surges. GitHub. https://github.blog/news-insights/octoverse/octoverse-2024.