542 AIDevSet Project

YUE WANG[*], University of British Columbia at Okanagan, Canada
YUSEN RONG[†], University of British Columbia at Okanagan, Canada

ACM Reference Format:
Yue Wang and Yusen Rong. 2025. 542 AIDevSet Project. 1, 1 (December 2025), 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

With the rise of artificial intelligence (AI) in recent years, there has been a lot of discussion about if and how soon AI can replace software engineers in most companies. Historically, the code generated by AI can be questionable, especially if the language is relatively new or not as popular. Several analysis will need to be done on the capabilities of AI agents before even broader adoption by companies, and is probably a yearly exercise for every company to evaluate effectiveness and performance. In this report, we will use some of the AIDev datasets to analyze some performance and adoption metrics to see where the state of AI agents is currently. The GitHub project for this report is located at https://github.com/yueywangse/542Project.

2 Analyzing of AI Agent Usage VS Coding Language

The first question that comes to mind is which coding language uses AI agent Pull Requests more often? Is there a specific tendency to adopt AI agents more in some languages vs others? This data is very interesting and might give us some insight on what language AI agents work best, where they need improvement, and if companies that utilize a certain language should start looking at adopting AI agent usage if they have not already.

2.1 Methodology

To determine the frequency of AI agent usage per language, we will mainly use two of the AIDev datasets, pull_requests and repository. We first checked that there are no Na values inside each of the repositories so that we can combine the two based on the repo_id column, unfortunately we found that around 37 out of 2807 repositories do not have data on what language they use, and with such a low number affected and no reliable way to collect that data ourselves, we simply discarded those rows of data. After the merge we confirmed that 183 of the 33596 rows of pull request data was also lost, which is a small number as well and was acceptable to us. With the cleaned data, we can then calculate and plot the frequencies of repos per language, the frequencies of prs per

[*]General Report structure, wrote Introduction and section 2 and subsections, wrote overviews for section 3 and 4, wrote code for section 2 and subsection, proof reader
[†]Wrote methodology and analysis for section 3 and 4, wrote code for section 3 and 4 and subsections

Authors' Contact Information: Yue Wang, yueywang@student.ubc.ca, University of British Columbia at Okanagan, Kelowna, British Columbia, Canada; Yusen Rong, , University of British Columbia at Okanagan, Kelowna, British Columbia, Canada.

ACM XXXX-XXXX/2025/12-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

language, and finally divide the two statistics to get average number of prs per repo per language to estimate language adoption.

2.2 Results

The generated frequency graphs are as follows.

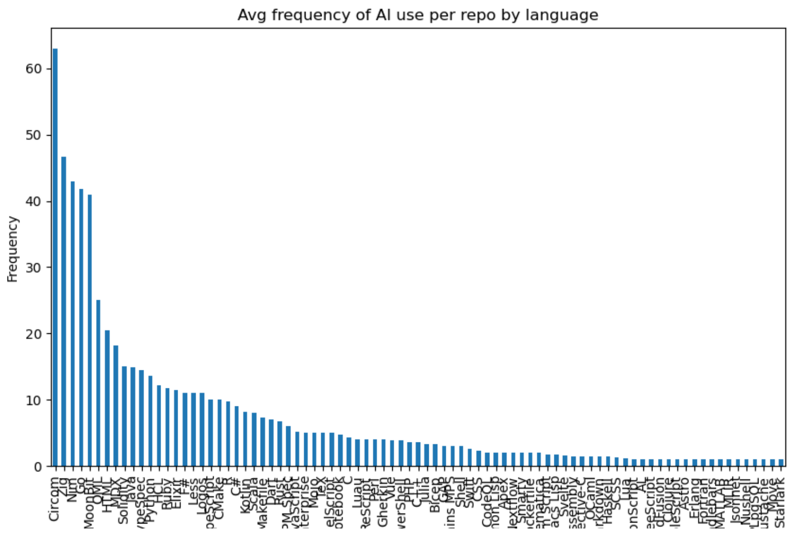

Fig. 1. Frequency Graph Depicting Average Number of AI Agent Prs per Repo by Language

2.3 Analysis

The final Fig. 1 above shows the average amount of AI agent pull requests per repo separated by language. This can serve as an estimate of AI agent adoption per language for consideration, and might help a company decide whether they should follow suit/continue on their path of AI integration if they also use the same language. Do note however that this graph can be misleading, as the language with the largest adoption of AI agents, Circom, is used in an extremely small number of repos, so it might just be the effect one one person who is really into AI adoption. We would recommend only consulting the results for the more popular programming languages and not all of them.

3 Analyzing Change Complexity of Different AI Agents

In this question, we will study how various factors of change complexity of different AI agents affect the time to merge and acceptance rate of AI agent created pull requests. This can be useful to know what factors influence acceptance rate and merge times to be help a company decide which AI agent they might want to use or migrate to to best fit the company's goals.

3.1 Methodology

For this question, we study how the size and complexity of AI-generated pull requests relate to their probability of being merged. We work with the AIDev dataset at the pull-request level and focus only on PRs created by AI agents (OpenAI Codex, Copilot, Devin, Cursor, Claude Code).

We first construct a base pull-request table from pull_request.parquet by renaming the primary key to pr_id and the agent column to agent_name. We then define a binary outcome variable merged_flag, which is set to 1 if the PR has a non-missing merged_at timestamp and 0 otherwise. This serves as our measure of "acceptance": a PR is considered accepted if it is merged into the main branch.

Next, we build change-complexity features using the pr_commit_details.parquet table. For each pr_id, we aggregate file-level statistics to obtain:

- total_additions: total lines added across all files,
- total_deletions: total lines removed,
- files_changed: number of unique files touched,
- total_changes: total lines changed (sum of changes across files).

We then merge these complexity features back into the PR table. To reduce skew, we define log_total_changes = log(1 + total_changes). Missing counts are treated as zeros, which corresponds to PRs with very small or trivial diffs.

As an initial descriptive step, we compute per-agent summaries: number of PRs, mean and median total_changes, and the empirical merge rate (merged_flag mean). This shows how each agent behaves in terms of typical patch size and acceptance rate.

To formally quantify the relationship between complexity, agent identity, and merge outcomes, we fit a logistic regression model. The response is merged_flag (1 = merged, 0 = not merged). The predictors are:

- numeric features: log_total_changes, files_changed;
- categorical feature: agent_name, encoded via one-hot encoding.

We standardize numeric features with StandardScaler and encode agent_name with OneHotEncoder inside a single scikit-learn pipeline. The dataset is split into training and test sets using an 80/20 stratified split to preserve the merge/non-merge ratio. To evaluate model stability and out-of-sample performance, we use 10-fold stratified cross-validation and report the mean ROC AUC and its standard deviation, as well as the ROC AUC on the held-out test set. Finally, we extract the logistic regression coefficients and convert them into odds ratios to interpret the effect of complexity and agent identity on merge probability.

3.2 Result

Descriptively, the five AI agents show clear differences in both change size and acceptance rates. OpenAI Codex submits the largest number of PRs and tends to produce relatively small, focused changes: its median total_changes is on the order of tens of lines, and its merge rate is high (approximately 83%). In contrast, Copilot and Devin submit substantially larger changes on average, with higher median total_changes, but achieve lower merge rates (around 43% for Copilot and 54% for Devin). Cursor and Claude Code occupy a middle ground: they typically propose medium-to-large patches with merge rates between those of Codex and Copilot.

The logistic regression model that combines change complexity (log_total_changes, files_changed) and agent identity (agent_name) achieves a mean ROC AUC of about 0.716 (standard deviation roughly 0.004) under 10-fold stratified cross-validation. On a held-out 20% test set, the ROC AUC is approximately 0.717. This means that, given a randomly chosen merged PR and a non-merged PR, the model gives a higher predicted merge probability to the merged PR about 71–72% of the time. Overall accuracy on the test set is around 71%, with the model performing better for the merged class (precision ≈ 0.83, recall ≈ 0.75) than for the non-merged class.

Looking at the coefficients, log_total_changes has a negative effect on merge odds: its estimated odds ratio is roughly 0.8, meaning that a one-standard-deviation increase in log total changes reduces the odds of being merged by about 20%, holding agent identity and other factors fixed. The coefficient on `files_changed` is small and slightly positive (odds ratio close to 1.05), suggesting that once we know how many lines are changed, the number of files touched adds relatively little additional information.

Agent identity also remains important after controlling for change complexity. The indicator associated with OpenAI Codex has a strong positive coefficient, with an odds ratio of about 2.5, implying that Codex PRs have more than double the merge odds of an "average" encoded agent at the same complexity level. Cursor has a mild positive effect (odds ratio slightly above 1), while Claude Code is roughly neutral (odds ratio just below 1). By contrast, Devin and especially Copilot have negative coefficients, with odds ratios of about 0.66 and 0.41 respectively, indicating that their PRs are less likely to be merged than similarly complex Codex or Cursor PRs.

3.3 Analysis

The results suggest two main findings for change complexity and AI agents.

First, change complexity clearly matters. Larger, more complex PRs—as measured by `total_changes`—are less likely to be merged, even after controlling for which agent created them. The negative odds ratio for `log_total_changes` formalizes a pattern that many practitioners intuitively expect: big diffs are harder to review, more likely to break existing behaviour, and therefore more likely to be rejected or left unmerged. The very small effect of `files_changed` once total lines changed are known suggests that reviewers care primarily about the overall amount of change rather than the exact number of files it is spread across.

Second, agent identity remains an important predictor of merge success over and above complexity. OpenAI Codex PRs are both smaller on median and substantially more likely to be merged, while Copilot and Devin PRs are larger and have noticeably lower merge odds. Cursor and Claude Code sit in between. Since the model controls for change size, these differences cannot be explained purely by the fact that some agents produce larger patches. Instead, they indicate that maintainers respond differently to different agents' outputs, possibly because of perceived code quality, style, or stability of the changes.

From a practical perspective, this analysis provides a simple way for an engineering organization to think about adopting or migrating between AI coding agents. If a team cares primarily about maximizing the fraction of AI-generated PRs that get merged with minimal human friction, then an agent that tends to produce smaller, more focused edits with higher conditional merge odds (such as Codex in our dataset) appears more attractive. If a team instead wants aggressive refactors or large-scale changes, they should expect lower acceptance rates and may need to invest more in code review and testing workflows.

Finally, we note that this question focuses on merge probability as our main outcome. Time-to-merge is also an important part of developer productivity, but it depends strongly on human review dynamics. In the next research question, we explicitly model review intensity (number of reviewers, timing of first review, and review activity) to see how human oversight interacts with agent identity and change complexity to shape both merge decisions and review effort.

4 Analyzing Review Intensity of Different AI Agents

For the final question, we will analyze how human scrutiny changes based on which AI agent constructed the pull request. We can measure the number of reviewers, total review comments, and time to first review for each AI agent to assess their performance compared to each other. This can also be helpful to companies deciding which AI agent they should work with.

WIP we're working on the calculations for this question and will update the sections along with a conclusion for the final copy.