Towards Understanding AI-Agent Traces in Open-Source Projects

Miguel Romero-Arjona, Ana B. Sánchez, and Sergio Segura

SCORE Lab, I3US Institute, Universidad de Sevilla, Seville, Spain
{mrarjona,anabsanchez,sergiosegura}@us.es

Abstract. AI agents are increasingly being integrated into software development environments, reshaping how open-source projects are designed, implemented and maintained. Despite their growing adoption, empirical research examining their influence is constrained by the difficulty of reliably identifying agent involvement within development platforms. In this paper, we explore the methodological challenges of detecting AI-agent traces in GitHub pull requests, where multiple agents may interact with varying levels of visibility. We formalise the detection problem and analyse three categories of observable agent usage signals—account-level indicators, branch naming conventions, and textual attribution—along with their individual limitations. Based on this analysis, we argue that no single observable signal is sufficient. Instead, robust detection requires integrating complementary sources of evidence and making underlying assumptions explicit, thereby enabling comparability and reproducibility across empirical studies.

Keywords: AI agents · GitHub · Pull requests

1 Introduction

AI agents are rapidly becoming embedded in modern software development workflows. They increasingly support tasks such as code generation, refactoring, documentation writing, and automated review, reshaping how developers produce and maintain software [2,4]. In open-source ecosystems, and particularly on GitHub, this shift creates an opportunity to study how AI agents influence productivity, collaboration, and code quality at scale. Open-source platforms expose rich traces of development activity, making them attractive environments for empirical software engineering research. However, such analyses depend on reliably determining *when* and *how* AI agents participate in the development process [3,5].

Despite the intended transparency of open-source development, detecting agent involvement remains challenging due to the heterogeneous ways in which agents are used [1]. In some cases, agents operate through clearly identifiable accounts, pushing commits or posting reviews under dedicated tool identities. In many others, agent involvement is not observable at the account level. For example, an agent may propose changes in an IDE or chat interface, after which a

developer manually applies and commits the modifications under their own identity. As a result, observable traces often provide only a partial, and potentially distorted, view of the underlying development process.

This paper frames the detection of AI-agent usage as a methodological challenge in empirical software engineering. We adopt GitHub pull requests (PRs) as our unit of analysis because they concentrate multiple forms of observable interaction (e.g., code changes, discussions, review activity). Specifically, we formalise the problem of detecting agent involvement (Section 2), identify observable signals of AI-agent usage and their associated limitations (Section 3), and discuss implications and future directions for building more robust detection methods (Section 4).

2 Problem

On GitHub, AI-agent usage is often manifest through PRs. Given instructions (e.g., prompts or issues), an agent can generate artefacts or take actions across the development lifecycle, such as proposing code changes or updating documentation. The key challenge is whether agent participation leaves traces that can be consistently attributed across repositories. Figure 1 shows how multiple agents can interact within a single PR[1], each with varying levels of visibility. In this example, the PR was opened by OpenAI Codex, identifiable through its characteristic branch-naming convention (item ①). The PR description was subsequently refined by Sourcery (item ②). Although Codex generated the underlying code changes, the corresponding commit was authored under a human account (item ③). Additionally, two other agents participated in the review process (item ④).

The challenge of detecting agent traces also depends on how detailed the study needs to be. A binary classification of agent involvement (involved vs. not involved) is simple and scalable, but hides distinctions that often matter most in practice, such as whether a contribution is agent-only or the result of human-agent collaboration. Finer-grained formulations (e.g., multi-class or multi-label schemes) better capture these interaction patterns and enable richer analyses, yet they require clearer operational definitions and stronger evidence to support consistent, reproducible detection.

3 Signals for Detecting Agent Usage

Detection signals can be categorised according to the part of the GitHub workflow from which they originate. **Account-level** signals are the most straightforward; if the PR author or any participant uses a dedicated agent identity, involvement is easier to infer. The main difficulty lies in determining whether an account is agent-operated. Although the GitHub API distinguishes between `bot` and `user` accounts, the `bot` value encompasses both AI agents and traditional

[1] `https://github.com/Bryan-Roe-ai/semantic-kernel/pull/786`

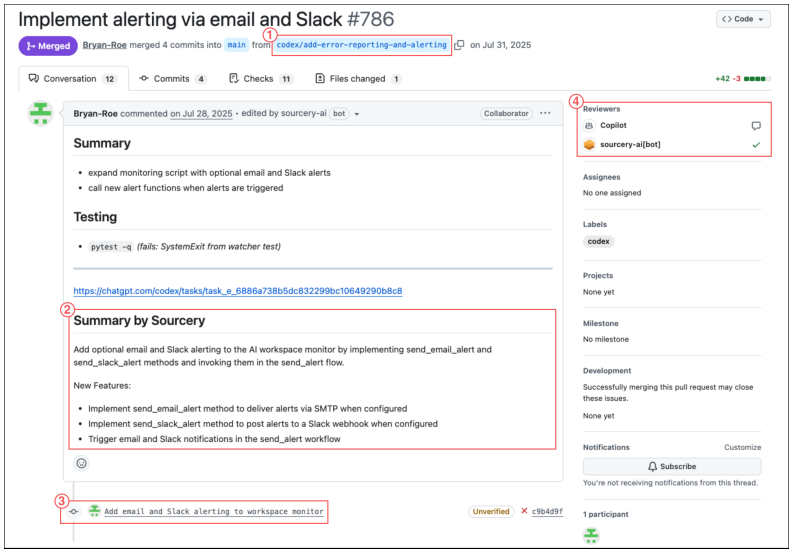

Fig. 1. Example of AI-agent traces within a GitHub pull request.

automation tools (e.g., dependency update bots), while even some AI agents can appear as regular `user` accounts (e.g., Claude[2]). Detection based on static allowlists of known agent usernames can achieve high precision but suffers from low recall and degrades as new tools emerge.

Branch naming conventions provide another potential source of evidence. Some agent tools generate branches using recognisable prefixes or templates that embed the tool name (e.g., `codex/<TEXT>` for Codex-generated PRs[3]). When present, such conventions may reveal agent involvement even when the PR is opened through a human account. However, these patterns are neither standardised nor stable. They are often configurable, may change across tool versions, and can collide with human naming conventions. As a result, detectors relying on branch names risk both false positives and temporal degradation.

Textual attribution in PR descriptions and commit messages constitutes a third class of observable signal. Many workflows encourage explicit attribution through phrases such as "Generated with Claude Code"[4], "Suggested fixes powered by Copilot Autofix"[5], or co-authorship trailers[6]. Nevertheless, free text is inherently noisy. Templates are reused across repositories, wording varies across languages, and attribution may be incomplete, exaggerated, or omitted altogether (e.g., for social or legal reasons). Keyword-based detection is also vulner-

[2] `https://api.github.com/users/claude`
[3] `https://github.com/gilad-tsur/ARC-AGI-2/pull/1`
[4] `https://github.com/x81k25/reel-driver/pull/18`
[5] `https://github.com/paufregi/GarminConnectFeed/pull/153`
[6] `https://github.com/getsentry/sentry/pull/93402`

able to negation (e.g., "not generated with") and historical references (e.g., "we used X last year"). Therefore, high-coverage textual heuristics require corroborating evidence to avoid substantial false-positive rates.

Overall, no single signal provides universally reliable evidence of agent usage. Robust detection approaches must combine multiple sources of evidence, document the assumptions underlying their design, and acknowledge that certain forms of agent involvement will remain observationally indistinguishable from purely human activity.

4 Conclusions

Detecting AI-agent usage in open-source repositories cannot be reduced to a single heuristic because agent participation is heterogeneous and frequently intertwined with human activity. We argue that identification strategies should integrate multiple signals (e.g., account-level information, branch conventions, and textual attribution) and make their assumptions explicit to ensure comparability and reproducibility across studies. As future work, we plan to develop and evaluate a methodology that systematically integrates these signals to improve the reliability of agent detection.

Acknowledgments. This work is a result of grant TED2021-131023B-C21, funded by MCIN/AEI/10.13039/501100011033 and by European Union "NextGenerationEU/PRTR"; grant DGP_PRED_2024_00262, funded by the Junta de Andalucía/CIIU and the FSE+; and project PID2024-156482NB-I00, funded by MICIU/AEI/10.13039/501100011033 and by the FSE+. This work is also supported by the Spanish Ministry of Science and Innovation under the Excellence Network AI4Software (Red2022-134647-T).

References

1. Agarwal, S., He, H., Vasilescu, B.: AI IDEs or Autonomous Agents? Measuring the Impact of Coding Agents on Software Development. arXiv preprint (2026). https://doi.org/10.48550/arXiv.2601.13597
2. Batole, F., OBrien, D., Nguyen, T.N., Dyer, R., Rajan, H.: An LLM-Based Agent-Oriented Approach for Automated Code Design Issue Localization. In: 2025 IEEE/ACM 47th International Conference on Software Engineering (2025). https://doi.org/10.1109/ICSE55347.2025.00100
3. Li, H., Zhang, H., Hassan, A.E.: The Rise of AI Teammates in Software Engineering (SE) 3.0: How Autonomous Coding Agents Are Reshaping Software Engineering. arXiv preprint (2025). https://doi.org/10.48550/arXiv.2507.15003
4. Nguyen, M.H., Phan Chau, T., Nguyen, P.X., Bui, N.D.Q.: AgileCoder: Dynamic Collaborative Agents for Software Development based on Agile Methodology. In: 2025 IEEE/ACM 2nd International Conference on AI Foundation Models and Software Engineering (2025). https://doi.org/10.1109/Forge66646.2025.00026
5. Watanabe, M., Li, H., Kashiwa, Y., Reid, B., Iida, H., Hassan, A.E.: On the Use of Agentic Coding: An Empirical Study of Pull Requests on GitHub. ACM Trans. Softw. Eng. Methodol. (2026). https://doi.org/10.1145/3798166