A Domain-Level Empirical Characterization of AI-Authored Pull Requests in Open-Source Software Engineering

Kevin Jie[*]
University of Waterloo
Waterloo, Ontario, Canada
kevin.jie@uwaterloo.ca

Xiangrui Ke[*]
University of Waterloo
Waterloo, Ontario, Canada
x3ke@uwaterloo.ca

Zhiheng Lyu[*]
University of Waterloo
Waterloo, Ontario, Canada
zhiheng.lyu@uwaterloo.ca

Abstract

AI coding agents are rapidly reshaping software development practices, yet the research community still lacks a comprehensive understanding of how these agents collaborate with human developers in real-world projects. To address increasingly contributing pull requests (PRs), yet we still lack concrete evidence about how their effectiveness varies across real-world software domains. To narrow this gap, we conduct an in-depth empirical study on using AIDev, the first open large-scale dataset of agent-authored pull requests (Agentic-PRs). To support large-scale analysis, we employ a large language model (LLM) to annotate PR domains and demonstrate that its labels closely approximate human judgment. We build a 15-category domain taxonomy and evaluate whether LLM-based domain annotation is reliable. Applying the taxonomy to 2,000 Agentic-PRs, we observe substantial domain differences: localized changes such as language, observability, documentation, and frontend show high merge rates about 70–80% and short review cycles around 20–40 hours, while domains involving cross-component dependencies or deeper semantics, e.g. compute-ML, system, data-processing, performance, exhibit markedly lower acceptance frequently below 40% and long-tailed review delays with a mean around 70–100 hours. These results indicate that the reliability of AI-generated PRs is domain-contingent, shaped by integration risk and verification cost. Our study provides evidence on where current agents perform well, where they struggle, and why certain domains remain challenging for AI–human collaboration.

1 Introduction

The rapid integration of AI coding agents, such as GitHub Copilot [8], Claude Code [5], and Cursor [6], into modern software engineering workflows is fundamentally transforming how software is developed, reviewed, and maintained. While early AI assistants operated primarily in an autocomplete or co-pilot role [9, 13, 20] recent advances in automated software engineering and LLM–based development tools have led to the emergence of Agentic-PRs. Today's Autonomous Coding Agents are beginning to function as AI teammates who pull requests and engage in feedback loops, reshaping software development workflows. These agents autonomously propose code changes, generate tests, and fix bugs, leading to a growing proportion of Agentic-PRs in open-source context.

The research community is actively studying how AI coding agents propose code changes and what kinds of collaboration patterns is [2, 17]. However, the commuity still lacks a systematic and domain-aware understanding of the effectiveness of Agentic-PRs varies across different software domains, such as web development, data processing, system programming, and machine learning. Prior works on code review bots [1, 19, 24] have shown that automation can increase accept rates and alter collaboration patterns in open-source projects, while studies of LLM–based code review systems [4, 15] highlight both productivity gains and coordination challenges introduced by these tools. In contrast, our work addresses this gap by systematically characterizing how the effectiveness of Agentic-PRs vary across domains, thus linking LLM behaviour to the semantic and structural diversity of real-world dataset. This gap poses a critical limitation for both practitioners and researchers. With domain-specific evidence, we can: accurately assess which domain AI agents are strongest at, and understand the interaction dynamics between humans and agents across domains. In this paper, we analyze more than 2,000 PRs from the AIDev dataset [11] and made several contributions:

- We introduce a finer-grained semantic taxonomy that distinguishing characteristics of different domains and enables LLMs to achieve more accurate understanding and more effective assistance during downstream analyses.
- We integrate LLM into our annotation pipeline and systematically mitigate potential biases through human cross-validation and rigorous prompt-engineering procedures. These measures collectively ensure high-fidelity, reliable, and unbiased domain labels for downstream analysis.
- We adopt statistical methods to assess domain-specific advantages, by applying confidence-interval estimation to all critical statistics, our approach yields statistically solid and convincing of our findings.

2 Motivation

Software development is inherently shaped by its domain context, which determines not only the technical structure of the systems being built but also the collaborative and organizational practices surrounding them. The categorization can provides a structured view of how AI-agents are being used across different software development activities.

Different domains differ substantially in their architectural constraints, testing requirements, and code review. Consequently, the effectiveness of AI coding agents in assisting developers is unlikely to be uniform across these domains. For instance, an exploratory data science project may be more tolerant of AI-generated code patches that facilitate rapid prototyping, whereas system software projects often require strict adherence to performance, correctness, and security standards, resulting in a much more conservative review process. This domain sensitivity implies that the same Agentic-PR may be reviewed and merged in fundamentally different ways depending on its application context.

[*]Both authors contributed equally to this research.

Table 1: Domain taxonomy for Agentic-PRs.

ID	Category	Description
1	system	System related: io, file-system, memory, exception, underflow, overflow, serialization, path, symlink, encoding, datetime, unicode, container (runtime context), os-compat, networking
2	language	Programming language related: type, encoding, unicode, regex, parsing, linting
3	architecture	Design related: schema, design, refactor, model, OOP
4	data-processing	Data related: SQL, NoSQL, parquet, Spark, distributed data, batch, graph, algorithm
5	compute-ml	Machine-learning related: torch, transformer, DNN, RNN
6	compute-gpu	GPU related: OpenGL, CUDA, XLA, Triton, hw-accel, quantization
7	infra	DevOps and configuration: env, container, build, logging, config, packaging, cloud, docker, registry, k8s-controller, custom-resource, CI, CD, release-eng, rollback, pipeline, proxy
8	security	Security and vulnerability related: auth, vulnerability fix
9	observability	Observability related: metrics, tracing, telemetry, log
10	performance	Performance related: bug-fix, caching, compression, session
11	frontend	UI related: screen, DOM, view
12	document	Documentation related: docstring, clarification, explanation, annotation
13	testing-quality	Testing related: test coverage, unit test, integration test
14	api-design	API and backend related: API endpoint, REST, Swagger, OpenAPI
15	others	Not suitable for any of the above categories

A systematic understanding of these differences can reveal where AI agents currently perform well, where they struggle, and how their capabilities might be better aligned with domain-specific development practices. Beyond descriptive insight, such understanding provides an empirical foundation for improving both the design of coding agents and the strategies. This is particularly significant as AI coding agents transition from assistive tools to active collaborators in large-scale software engineering projects.

3 Related Work

3.1 AI for Software Engineering (AI4SE)

Recent papers on AI4SE [12, 16, 18] articulate the need for "trustworthy and synergistic" human–AI collaboration, emphasizing reliability, transparency, and socio-technical alignment as key challenges when embedding AI agents into developer workflows. The emergence of large language models has further accelerated AI4SE [7, 10, 21], with recent surveys on LLMs for SE (LLM4SE) mapping studies cataloguing hundreds of applications of code- and chat-based models to tasks such as code completion, program repair, documentation, test generation, and requirements analysis, and calling out open problems in evaluation, safety, and tool integration.

3.2 Empirical AI-Assisted Code Agent

Empirical studies of industrial AI coding tools further demonstrate their practical impact. Peng et al. [14] from Microsoft, for instance, reported that GitHub Copilot can significantly reduce task completion time and increase perceived productivity. Moreover, recent works [3, 22] analyzed developers' cognitive load and trust factors and showed that 30.7% of the commits generated by AI-agents were still unsatisfying, illustrating persistent concerns about correctness, security, and over-reliance. More recent works have explored category-level characterizations at the commit level [25] and PR-level classifications [23], as well as factors associated with PR acceptance, the literature lacks a domain-specific taxonomy with sufficiently fine-grained semantic resolution. Such a taxonomy is crucial for capturing meaningful cross-domain heterogeneity that coarse-grained categories fail to reveal. To address this gap, we adopt a statistically rigorous approach to quantify acceptance differences across our fine-grained domains and construct confidence intervals to assess the significance and robustness of these disparities.

4 Research Questions

We aim to understand which software domains LLM-based agents perform best in, and how domain-specific characteristics affect the acceptance of Agentic-PRs. Our analysis leverages existing domain tags derived from commit and repository metadata. The domain taxonomy is shower in Table 1. The selected categories are from an SE agent project done in a previous internship.

To enable our analyses, we first require a systematic domain annotation of each pull requests. We carefully curated a set of 15 domains designed to comprehensively capture the semantic space of Agentic-PRs. Manually annotate domains through human interpretation is highly labor-intensive and does not scale effectively. Given the strong semantic understanding capabilities demonstrated by recent powerful LLMs, we investigate whether advanced LLMs can assist in domain annotation. However, the reliability of LLM-based domain labeling for Agentic-PRs remains largely unknown.

To address these gaps and to provide a data-driven characterization of domain-specific patterns in Agentic-PR behaviour, we formulate the following research questions:

RQ1: How accurately can LLMs identify the software domains of Agentic-PRs?

This question evaluates whether LLM-based annotation pipelines can reliably assign multi-domain semantic labels at scale. Since our downstream analyses depend on the fidelity of domain classification, establishing the accuracy, limits, and failure modes of LLM-driven labeling is a foundational requirement for the study.

RQ2: Which domains exhibit higher Agentic-PR accept rates? What kind of characteristics most strongly influence their likelihood of being merged?

Acceptance or rejection reflects human trust, perceived quality, risk tolerance, and domain-specific integration complexity. By analyzing acceptance rates across 15 semantic domains and quantifying their statistical significance, we aim to uncover domain-level structural factors that affect the merge outcomes of AI-generated contributions.

RQ3: What is the distribution of review times for Agentic-PRs , and which factors would cause the long-tail review delays if any?

Review latency provides insight into the collaborative burden introduced by Agentic-PRs. This question investigates how review durations vary across domains, why some domains experience substantially slower cycles, and what technical or socio-technical factors contribute to extended review delays.

Collectively, these research questions enable a domain-aware empirical understanding of the behaviour, limitations, and collaborative implications of AI-authored pull requests, forming the basis for the analyses presented in the subsequent sections.

5 Methodology

5.1 Dataset

We will use AIDev [11], the first large-scale, openly available dataset capturing Agentic-PRs from real-world GitHub repositories:

- Scale: 932,791 Agentic-PRs
- Breadth: 116,211 repositories and 72,189 developers, across five AI agents: Claude Code, Cursor, Devin, GitHub Copilot, OpenAI Codex
- Depth: 33,596 curated Agentic-PRs from 2,807 popular repositories (over 100 stars), enriched with comments, reviews, commits, and related issues

5.2 PR States and Data Preprocessing

To characterize the lifecycle of Agentic-PRs and prepare the dataset for analysis, we first normalize the state information associated with each PR. Every PR in our dataset records a creation timestamp, as well as the timestamp at which it was closed or merged when applicable. Based on these event logs, we derive their final states, distinguishing between *open, closed*, and *merged*. Following prior work [23], we treat merged PRs as the subset of closed PRs that have been successfully accepted, and we define the **accept rate** as the percentage of closed PRs that end in a merge state, and closed-but-unmerged as **failures**.

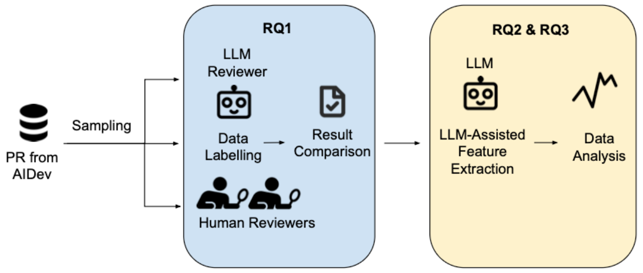

Figure 1: Overview of the Study Design.

In addition, we compute the review time for each PR by measuring the elapsed time between its creation and closure timestamps. Review time is expressed in hours and reflects the duration of the end-to-end review process.

5.3 Tools

We use **GPT-5** as the primary model for labeling **2,000 representative Agentic-PRs**, chosen for its balance between cost and accuracy. We choose LLM because: first, the large sample size makes manual labeling difficult; second, LLM is capable of handling complex classification tasks; and third, because LLM is pre-trained and has strong performance, we do not need to train a new model. While an alternative to LLM is to use a traditional machine learning model for labelling (classification task), LLM remains the best choice.

Prompts for domain classification are defined in structured templates (Appendix Table 3), each returning 1–3 deterministic tags for reproducible parsing.

All experiments are implemented through a reproducible Python pipeline, with metrics including precision, recall, F1.

5.4 Study Design

After data preprocessing, we sampled 100 tags to construct a gold-standard evaluation subset. Unlike independent annotation, the two human annotators jointly examined all items and established the gold labels through *consensus*. Whenever initial judgments diverged, the annotators conducted thorough discussions until full agreement was reached. This procedure ensured that the final gold-standard labels reflect a rigorously validated and mutually endorsed human understanding.

To improve the LLM-assisted annotation process, we split the 100 samples into two equal parts. The first 50 samples were used as a development set to identify bad cases and iteratively refine our prompting strategy. Based on model feedback, we designed three alternative prompt versions and compared their performance on this development subset. The best-performing prompt was then fixed and evaluated on the remaining 50-sample held-out set to validate its generalization benefits.

After selecting the final prompt, we applied it directly to GPT-5 to annotate all 100 samples in the evaluation subset. The model-produced labels were then compared with the human-consensus gold labels to assess alignment quality. Table 2 reports the following commonly used metrics for multi-label classification:

- **EM (Exact Match):** The proportion of samples for which the predicted label set is exactly identical to the gold-standard label set.
- **Precision (macro):** For each label, we compute precision independently and then average across all labels.
- **Recall (macro):** Similarly, recall is computed per label and macro-averaged across labels.
- **F1 (macro):** The harmonic mean of macro Precision and macro Recall.

Originally, we also decided to include human-human and human-model agreements by Krippendorff's α. However, since we have too many categorical labels and partial agreement exists, we decide to use traditional classification ML task metrics.

Table 2: Evaluation of GPT-5 annotation quality against human-consensus gold labels (100-sample multi-label task).

EM	Precision (macro)	Recall (macro)	F1 (macro)
0.40	0.710	0.730	0.692

5.5 Mitigating State Imbalance Through Balanced Sampling

Our goal is to examine how PRs in the dataset relate to the "state" of a pull request (open vs closed). However, the distribution of states in this dataset is highly skewed: more than 92% of pull requests are in the closed state. Such severe class imbalance implies that, within each category, closed pull requests numerically dominate the open ones. If any analysis or predictive model built directly on this distribution would be biased toward the majority class, making it trivial to achieve high apparent performance by almost always predicting "closed" while providing little insight into the characteristics gaining from data in "open" state.

To address this imbalance, we construct a balanced working sample by first drawing an equal number of closed and open pull requests and then conducting our classification and comparative analyses on this balanced subset. This sampling strategy reduces the risk of majority-class dominance and allows us to more clearly observe how different pull request categories are associated with each state. Importantly, we perform this balancing through random sampling, which preserves the internal heterogeneity of both the open and closed classes and avoids introducing systematic bias in favor of any specific project or category. As our primary objective in this part of the study is to characterize discrepancy between states, using a balanced sample is methodologically appropriate: it improves the stability and interpretability of the results without altering the underlying descriptive statistics.

5.6 Excluding Newly Created PRs to Avoid Right-Censoring Bias

To ensure the validity of our empirical analysis, we distinguish between **long-term open** pull requests and those that have been opened only recently. While some PRs remain open because they are unlikely ever to be resolved-a signal of latent process or quality issues-newly created PRs often still have a substantial chance of being resolved successfully. Including these very recent PRs in our dataset would introduce right-censoring bias, as their eventual state is not yet observable.

We distinguish PRs that are plausibly still *in-progress* from those that have effectively failed to reach close state. Consequently, we exclude all PRs opened very recently of data collection to mitigate bias introduced by uncertainty of the final state and to support more reliable analyses.

6 Results

6.1 The Accuracy of LLM-assistant Annotation

To evaluate whether LLMs can reliably classify the software domains of Agentic-PRs, we conducted a controlled validation in Sec 5.4 using a human-annotated subset of N = 100 pull requests. Overall, the LLM achieved an relative positive accuracy showed in table 2, even though the model rarely reproduced the full multi-label domain set exactly as assigned by human annotators. These results suggest that while strict EM scoring underestimates performance in multi-domain settings, the LLM captures the dominant semantic domains of Agentic-PRs with reasonable fidelity.

To further ensure labelling reliability, we performed a cross-checking procedure in which two human annotators independently reviewed a random subset of LLM-generated labels. Interestingly, during cross-validation, we also identified several inconsistencies in the human-labelled annotations, initially resulting in only 0.73 agreement, suggesting that human labelling is itself subject to non-trivial bias and error. The LLM's annotations showed a precision of 0.71 with human consensus, confirming that the model's labels are sufficiently correct. Based on this level of inter-rater agreement, we conclude that LLM-generated domain labels are reliable enough to support further large-scale annotation, offering a high-fidelity approximation of human judgment with substantial efficiency.

6.2 Accept Rate across Domains and Influencing Factors

With verified LLM annotation process, we examined domain-specific acceptance patterns across 2,000 Agentic-PRs covering 15 semantic domains. Figure 2 shows the counts of all our data. We exclude the *compute-gpu* domain since its data size is too small to produce statistically meaningful result; the corresponding confidence intervals are extremely wide. We therefore exclude this domain from cross-domain comparison. Figure 3 – Figure 5 summarizes both close and merge rates with 90% confidence intervals and hence, the accept rate for each domain.

We find substantial domain-level differences in accept rates of Agentic-PRs. High-acceptance domains such as *language, observability, documentation, and frontend* typically involve localized, low-risk modifications with predictable semantics and straightforward modification. In contrast, domains characterized by complex cross-module dependencies or high correctness risk, such as *compute-ml, system, performance, and data-processing*, show lower and more variable acceptance. These results suggest that the likelihood of an Agentic-PR being merged is strongly shaped by a domain's semantic complexity, integration risk, and reviewer confidence in validating correctness.

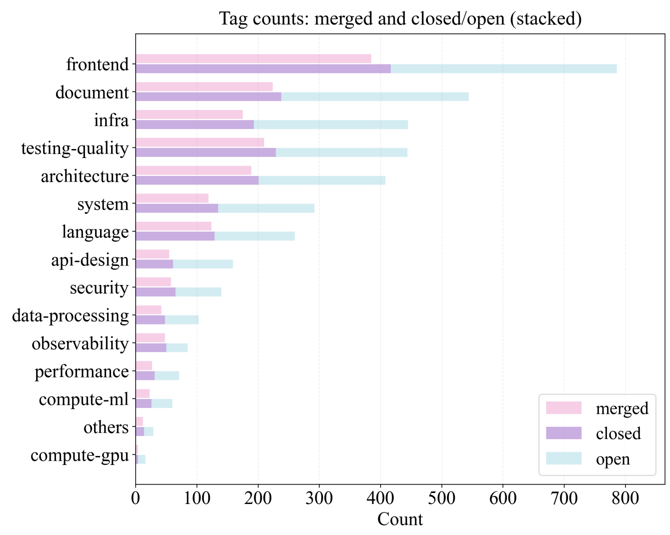

Figure 2: States Counts in 2K Balanced Tags

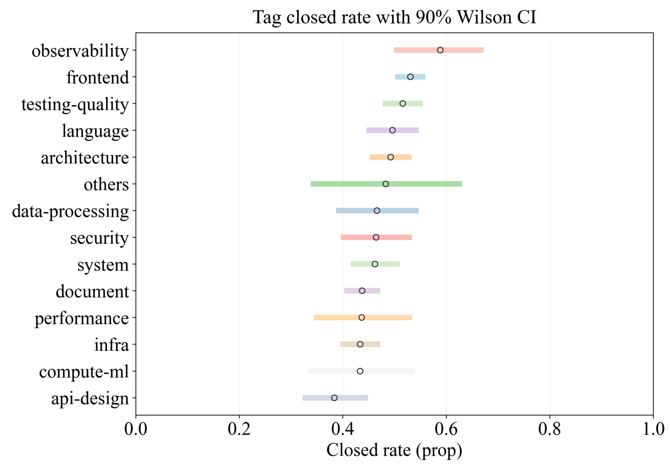

Figure 3: Close Rates Across Domains Under a 90% Confidence Threshold

6.3 Review Time across Domains and Influencing Factors

Furthermore, we analyzed *review times* for closed Agentic-PRs across domains. Figure 6 The distribution clearly exhibits a long-tail structure: while many PRs are reviewed within between 20 to 40 hours, several domains contain PRs whose mean review time exceeds 70–100 hours, with large standard deviations indicating substantial variability.

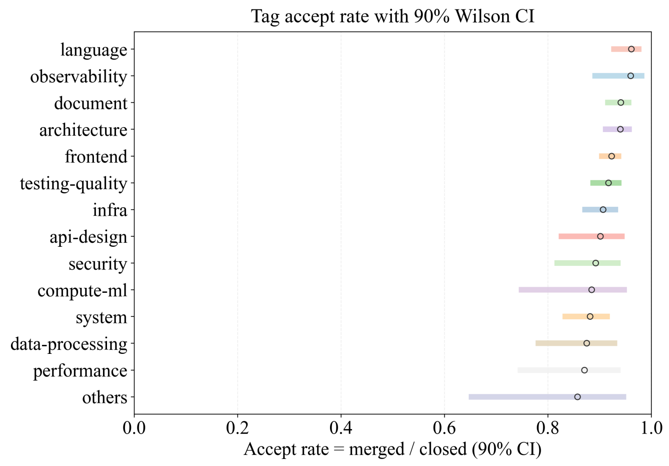

Figure 4: Accept Rates Across Domains Under a 90% Confidence Threshold

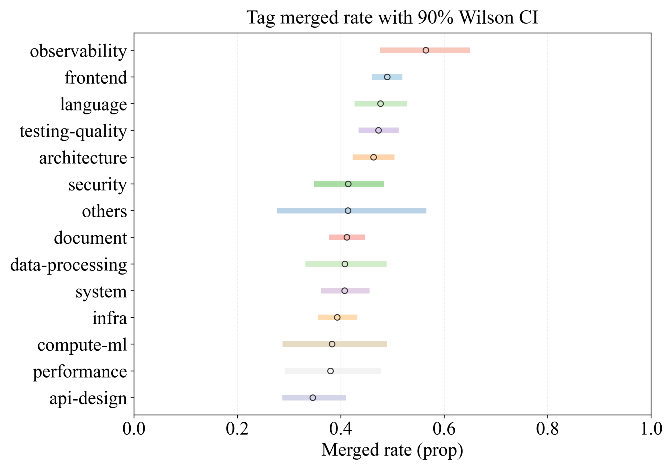

Figure 5: Merge Rates Across Domains Under a 90% Confidence Threshold

While most PRs are reviewed within 20–40 hours, domains with high semantic complexity or risk-such as *security, data-processing, compute-ml, and system*, show substantially longer and more variable review delays. These delays are driven by deep semantic dependencies, high correctness risk, multi-component interactions, and expertise bottlenecks. In contrast, domains with localized or syntactically bounded changes, e.g. *language, observability, testing-quality*, experience consistently short review cycles.

7 Meta-discussion

Our empirical analysis offers a roadmap for identifying where these gaps lie and which domain characteristics matter most in designing AI-augmented development tools in the future.

A broader implication of our findings is that domain-level analysis can serve as a foundation for improving the next generation of

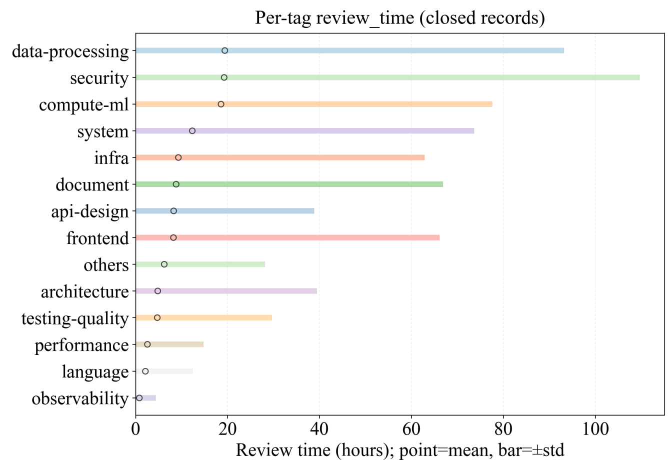

Figure 6: Average Review Time Across Domains

AI agents in software engineering. Integrating domain awareness into AI agents: whether through specialized reasoning modules, schema validation mechanisms, or explicit design guidelines, holds the potential to substantially improve PR quality and reviewer confidence.

To further understand how domain factors influence PR outcomes, we analyzed the semantic characteristics of ***rejected*** Agentic-PRs in Appendix 7. Focusing on technical terminology after removing common stopwords like "and", "or", we observed that terms such as *build, npm, hosting, mcp, layout, environment, and yaml* appear disproportionately often in rejected contributions. These terms correspond to domains where modifications require precise configuration semantics, environment-sensitive reasoning, or careful coordination across multiple components, remaining challenging for current AI systems. This suggests that the roots of rejection are not arbitrary but linked to structural and systematic complexities inherent to specific technical domains.

Taken together, these insights point toward a promising direction: domain-aware AI agent design. By understanding which domains introduce the highest cognitive load, risk, or dependency complexity, researchers and practitioners can better guide AI agents to produce more robust, trustworthy, and contextually appropriate PRs.

8 Threats to Validity

Our study faces several non-trivial threats to validity that should be considered when interpreting the results.

8.1 Construct Validity

The accuracy of our domain-level analysis depends heavily on the quality of LLM-based domain annotation. Although we designed structured prompts, employed iterative refinement, and validated model outputs against a human-annotated gold set, LLM predictions may still embed latent biases or misinterpret subtle domain cues. Multi-label domain assignments in particular may underrepresent secondary or less explicit semantic signals in PR descriptions. Additionally, the PR text field may not fully capture the technical intent when commit messages or inline code changes contain additional context.

8.2 Internal Validdty

While we applied balancing techniques to mitigate extreme state imbalance, residual confounding factors may persist. Differences in accept rates may reflect unobserved variables such as repository governance style, maintainer expertise, project maturity, or varying familiarity with specific AI agents. Review time is also affected by factors unrelated to domain complexity, such as reviewer availability, batching of review workloads, or project-specific triaging rules.

8.3 External Validity

Our findings are grounded in the AIDev dataset, which contains PRs produced by specific AI-agents within open-source environments. The results may not generalize to enterprise codebases, proprietary development workflows, or emerging agent architectures with different capabilities. The subset of 2,000 PRs used for deep analysis still captures only a portion of the broader 900K+ PR corpus. Finally, agent behaviors and developer attitudes evolve quickly; temporal drift may limit the long-term relevance of some observed patterns.

9 Future Work

Our study exposes several promising directions for further research on AI-assisted software development and Agentic-PRs. Here we briefly discuss three points.

Richer Domain Semantics and Multi-Granular Taxonomies. Future work could expand the domain taxonomy to incorporate architectural layers, dependency structures, or fine-grained risk profiles. Integrating static analysis or dynamic program analysis may yield domain signals that surpass text-based classification.

Improving Agent Design Through Domain Awareness. Our results point toward the value of domain-sensitive agent architectures, e.g. agents equipped with environment simulators, dependency reasoning modules, or domain-specific verification routines. Future systems may dynamically adapt generation strategies based on domain risk or reviewer expectations.

Human–AI Collaboration Dynamics. Understanding how developers perceive, trust, or adapt their workflows around Agentic-PRs remains underexplored. Qualitative studies, controlled user experiments, and ethnographic observations could reveal collaboration patterns and friction points invisible in PR-level data.

10 Conclusion

We presented a domain-level empirical study of Agentic-PRs, showing that LLM-assistant domain labeling is reliable, accept rates vary significantly across domains, and review times follow a heavy-tailed pattern shaped by domain complexity. These insights reveal where AI-agent excel at and provide guidance for building more domain-aware, trustworthy AI agents for future software development workflows.

References

[1] Adam Alami and Neil A. Ernst. Human and machine: How software engineers perceive and engage with ai-assisted code reviews compared to their peers. In *18th IEEE/ACM International Conference on Cooperative and Human Aspects of Software Engineering, CHASE@ICSE 2025, Ottawa, ON, Canada, April 27-28, 2025*, pages 63–74. IEEE, 2025.

[2] Fraol Batole, David O'Brien, Tien N. Nguyen, Robert Dyer, and Hridesh Rajan. An llm-based agent-oriented approach for automated code design issue localization. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*, pages 1320–1332. IEEE, 2025.

[3] Moataz Chouchen, Narjes Bessghaier, Mahi Begoug, Ali Ouni, Eman Alomar, and Mohamed Wiem Mkaouer. How do software developers use chatgpt? an exploratory study on github pull requests. In *Proceedings of the 21st International Conference on Mining Software Repositories*, pages 212–216, 2024.

[4] Umut Cihan, Vahid Haratian, Arda İçöz, Mert Kaan Gül, Ömercan Devran, Emircan Furkan Bayendur, Baykal Mehmet Uçar, and Eray Tüzün. Automated code review in practice. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 425–436, 2025.

[5] Claude Code. Claude code | claude, 2025.

[6] Cursor. Cursor - the ai code editor, 2024.

[7] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. Large language models for software engineering: Survey and open problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*, pages 31–53, 2023.

[8] GitHub. Github copilot · your ai pair programmer, 2025.

[9] Robert Haase. Towards transparency and knowledge exchange in ai-assisted data analysis code generation. *Nat. Comput. Sci.*, 5(4):271–272, 2025.

[10] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review. *ACM Trans. Softw. Eng. Methodol.*, 33(8), December 2024.

[11] Hao Li. Aidev, 2025.

[12] David Lo. Trustworthy and synergistic artificial intelligence for software engineering: Vision and roadmaps, 2023.

[13] Mira Mezini. Ai-assisted programming: From intelligent code completion to foundation models: A twenty-year journey. Software Engineering 2025, 2025.

[14] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. The impact of ai on developer productivity: Evidence from github copilot, 2023.

[15] Nikitha Rao, Bogdan Vasilescu, and Reid Holmes. From overload to insight: Bridging code search and code review with llms. In Leonardo Montecchi, Jingyue Li, Denys Poshyvanyk, and Dongmei Zhang, editors, *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering, FSE Companion 2025, Clarion Hotel Trondheim, Trondheim, Norway, June 23-28, 2025*, pages 656–660. ACM, 2025.

[16] Shu-Ting Shi, Ming Li, David Lo, Ferdian Thung, and Xuan Huo. Automatic code review by learning the revision of source code. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4910–4917, 2019.

[17] Thiago Rocha Silva and Thomas Troels Hildebrandt. Human-centric hybrid-ai for no-code development. In Leonardo Montecchi, Jingyue Li, Denys Poshyvanyk, and Dongmei Zhang, editors, *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering, FSE Companion 2025, Clarion Hotel Trondheim, Trondheim, Norway, June 23-28, 2025*, pages 1520–1521. ACM, 2025.

[18] Dominik Sobania, Martin Briesch, Carol Hanna, and Justyna Petke. An analysis of the automatic bug fixing performance of chatgpt. In *2023 IEEE/ACM International Workshop on Automated Program Repair (APR)*, pages 23–30. IEEE, 2023.

[19] Michal Szczepanik. Enhancing hotspot detection with behavioral analysis and ai-assisted code review. In Ngoc Thanh Nguyen, Anh-Vu Dinh-Duc, Adrianna Kozierkiewicz, Sinh Van Nguyen, Manuel Núñez, Jan Treur, and Gottfried Vossen, editors, *Computational Collective Intelligence - 17th International Conference, ICCCI 2025, Ho Chi Minh City, Vietnam, November 12-15, 2025, Proceedings, Part I*, volume 16138 of *Lecture Notes in Computer Science*, pages 361–375. Springer, 2025.

[20] Kishanthan Thangarajah, Boyuan Chen, Shi Chang, and Ahmed E. Hassan. Context-aware codellm eviction for ai-assisted coding, 2025.

[21] Rosalia Tufano, Simone Masiero, Antonio Mastropaolo, Luca Pascarella, Denys Poshyvanyk, and Gabriele Bavota. Using pre-trained models to boost code review automation. In *Proceedings of the 44th international conference on software engineering*, pages 2291–2302, 2022.

[22] Miku Watanabe, Yutaro Kashiwa, Bin Lin, Toshiki Hirao, Ken'Ichi Yamaguchi, and Hajimu Iida. 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*, pages 375–380, 2024.

[23] Miku Watanabe, Hao Li, Yutaro Kashiwa, Brittany Reid, Hajimu Iida, and Ahmed E Hassan. On the use of agentic coding: An empirical study of pull requests on github. *arXiv preprint arXiv:2509.14745*, 2025.

[24] Mairieli Wessel, Alexander Serebrenik, Igor Wiese, Igor Steinmacher, and Marco A. Gerosa. Quality gatekeepers: investigating the effects of code review bots on pull request activities. *Empirical Softw. Engg.*, 27(5), September 2022.

[25] Qunhong Zeng, Yuxia Zhang, Zhiqing Qiu, and Hui Liu. A first look at conventional commits classification. In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering*, pages 2277–2289, 2025.

A Appendix

A.1 Prompt of Domain Classification

Please check Table 3

A.2 AI Usage Statement

We used AI tools in three ways. First, large language models assisted with language editing (grammar, wording, and minor rephrasing), while the conceptual content and conclusions were authored and verified by the researchers. Second, AI-assisted coding and data analysis were used to accelerate implementation and exploratory analysis; all generated code and results were reviewed and, when necessary, revised by the authors. Third, AI models were used to propose initial data labels, which were subsequently checked and corrected by human annotators. In all cases, AI outputs were treated as suggestions and were subject to human validation before inclusion in this work.

Table 3: Prompt of Domain Classification

You are a classification model that assigns technical tags to pull-request descriptions. Given the PR **text** field, output 1–3 tags representing the most relevant technical domains involved.

————————— **Domain Taxonomy** —————————

1. system - OS, filesystem, IO, memory, exceptions, underflow/overflow, serialization, path, symlink, encoding, datetime, unicode, container runtime, OS compatibility, networking
2. language - programming language correctness: types, encoding, unicode, regex, parsing, linting
3. architecture - design-level concerns: schema, refactor, modeling, OOP, system structure
4. data-processing - SQL/NoSQL, ETL, parquet, Spark, distributed data, batch, algorithmic workflows
5. compute-ml - ML/DL: torch, transformers, DNN/RNN
6. compute-gpu - GPU/CUDA/OpenGL/XLA/Triton, quantization, acceleration
7. infra - DevOps: env, containers, build, logging config, packaging, cloud, docker, registry, k8s, CI/CD, release
8. security - auth, permission, security fixes, vulnerabilities
9. observability - logging, metrics, tracing, telemetry
10. performance - perf fixes, caching, compression, throughput, latency
11. frontend - UI, DOM, rendering
12. document - documentation, explanation, annotation, docstrings
13. testing-quality - tests, coverage, reliability
14. api-design - API endpoints, backend interface design, REST, OpenAPI/Swagger
15. others - only if none above fits

————————— **Output Rules** —————————

1. Return 1–3 tags, comma-separated, all lowercase, no spaces, no quotes.
2. Choose the most specific tags possible.
3. If multiple domains apply, order by relevance.
4. No explanations or commentary.
5. Output format: `<tag1,tag2,tag3>`

————————— **Input Text** —————————

`{text}`

A.3 Word Frequency in Failed PRs

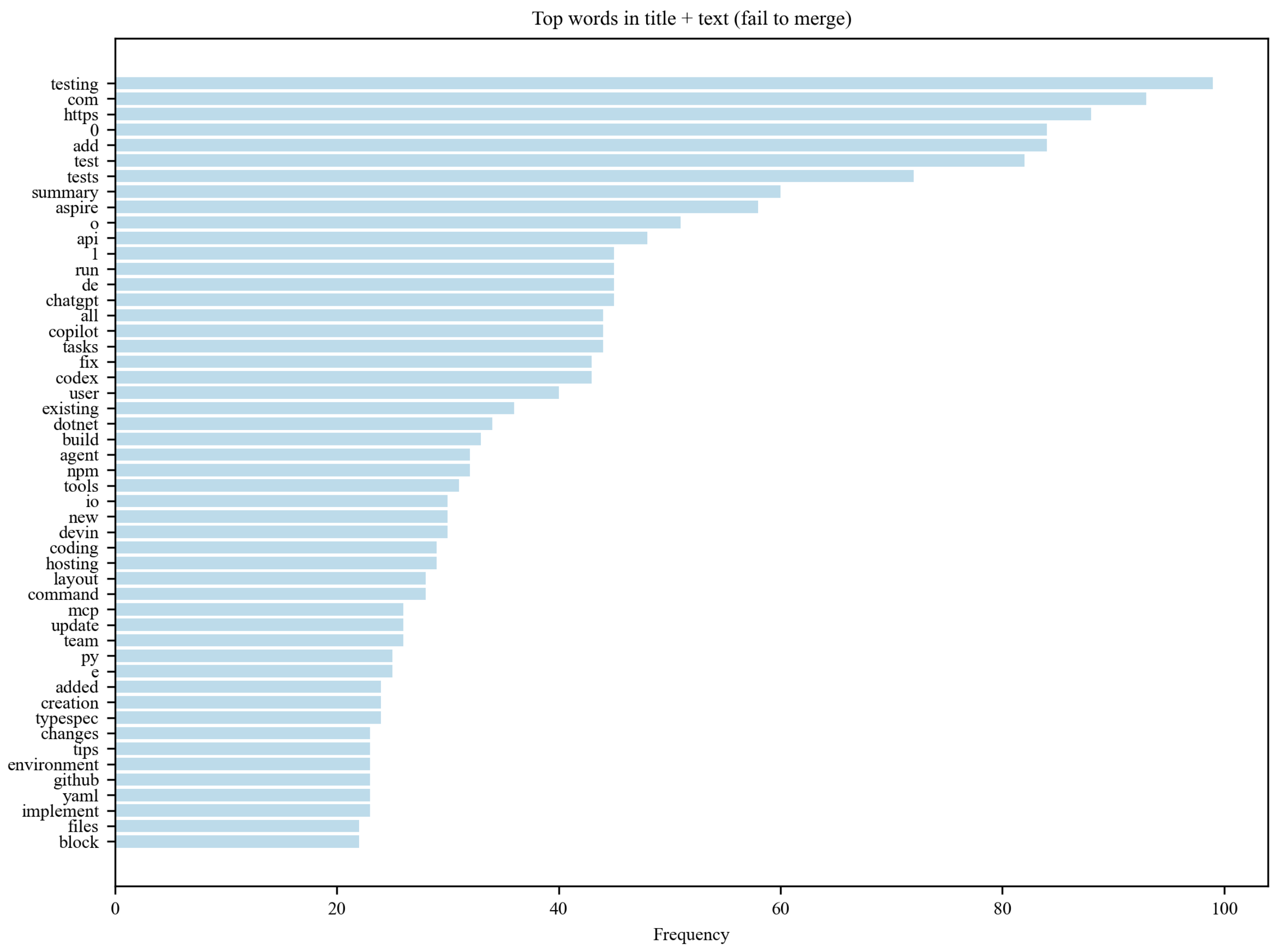

Figure 7: Terminology Word Frequency in Failed PRs(out of 2,000 data)