Understand Domains in Software Engineer: An Empirical Study on Agentic Pull Requests

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

1 Problem

The rapid integration of AI coding agents, such as GitHub Copilot [4], Claude Code [2], and Cursor [3], into modern software engineering workflows is fundamentally transforming how software is developed, reviewed, and maintained. These agents autonomously propose code changes, generate tests, and fix bugs, leading to a growing proportion of AI-generated pull requests (Agentic-PRs) in open-source context.

Despite this significant shift, the research community currently lacks a systematic and domain-aware understanding of how the effectiveness of Agentic-PRs varies across different software domains, such as web development, data analytics, system programming, and machine learning, and agents, such as GitHub Copilot and Claude Code. Prior work on code review bots [8] has shown that automation can increase merge rates and alter collaboration patterns in open-source projects, while studies of LLM–based code review systems [1] highlight both productivity gains and coordination challenges introduced by these tools. More recent analysis of Agentic-PRs [7] have explored their general acceptance and human–AI interaction dynamics. However, these investigations treat software projects as a uniform landscape, which largely ignoring the heterogeneous nature of software domains. In contrast, our work addresses this gap by systematically characterizing how the effectiveness and acceptance of Agentic-PRs vary across domains and code agents, thus linking LLM behavior to the semantic and structural diversity of real-world codebases. 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
- Understand the interaction dynamics between humans and agents across domains

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. Different domains differ substantially in their architectural constraints, testing requirements, code review rigor, and community norms. 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 perceived, reviewed, and merged in fundamentally different ways depending on its application context.

A systematic understanding of these differences is crucial: it 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 through which they are integrated into real-world software development workflows. This is particularly significant as AI coding agents transition from assistive tools to active collaborators in large-scale software engineering projects, amplifying their potential impact on code quality, productivity, and team dynamics.

3 Dataset and Tools

3.1 Dataset

We will use AIDev [5], 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

3.2 Tools

We use **DeepSeek-V3** as the primary model for labeling **5,000 representative Agentic-PRs**, chosen for its balance between cost and accuracy. A subset of 500 samples will be cross-validated using **GPT-5** and **Claude 4.5 Sonnet** to estimate model consistency and bias. 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 labeling (classification task), LLM remains the best choice.

To obtain a balanced and representative subset, we use **Qwen3-8B-Embedding** to encode commit messages, PR titles, and diffs. These embeddings support semantic clustering, retrieval-based few-shot prompting, and visualization of domain relationships. By selecting samples based on their embedding similarity rather than random choice or frequency, this method mitigates bias toward overrepresented patterns and yields a fairer, more evenly distributed dataset.

*Both authors contributed equally to this research.

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

All experiments are implemented through a reproducible Python pipeline, with metrics including precision, recall, F1, and cross-model agreement (κ).

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 (e.g., `api`, `parsing`, `exception`, `numerical`, `algorithm`, `networking`, `schema`, `config`, `serialization`, `cli`, etc.) derived from commit and repository metadata.

RQ1: How accurately can LLMs identify software domains of Agentic-PRs?
Goal: Build a scalable and reliable domain-labeling method for large datasets.
Method: Following [6], we will manually label a statistically significant subset with confidence 90%, margin ±5%, resulting about 271 samples and evaluate few-shot LLM classification on textual (README, commits) and structural (dependency graph, directory names) features. Precision, recall, and inter-domain confusion will be reported among tags such as `api`, `parsing`, and `networking`.

RQ2: Which domains and AI agents exhibit higher Agentic-PR acceptance rates and shorter review times?
Goal: Quantify domain-level and agent-level differences in PR success.
Method: Combine AIDev PR data with domain tags to compute acceptance rate, review latency, and comment density per domain. Control for repository popularity, PR size, and agent identity (e.g., Claude Code, Cursor, Copilot). Model acceptance likelihood using mixed-effects regression:

$$P(\text{accept}) = f(\text{domain}, \text{agent})$$

Domains such as `api`, `cli`, and `config` are expected to have higher success, while `algorithm`, `security`, and `networking` are likely more conservative.

RQ3: How do different AI agents behave across domains?
Goal: Identify cross-domain performance patterns of major coding agents.
Method: Compare domain-specific merge rates, review latency, and comment density across agents (e.g., DeepSeek-V3, Claude Code, Cursor, Copilot). Analyze whether agents show consistent strengths in similar domains, using variance decomposition and correlation analysis across domain clusters.

5 Plan

- Firstly, we will manually label a sample subset from the original dataset as a validation set. We will annotate the domains each Agentic-PR belongs to. Then we will use DeepSeek-V3 to classify the same dataset and measure its performance by precision, recall, and F1 scores. This task can answer RQ1 and is expected to be done by **Nov 9th**.
- Secondly, we will randomly select a subset of the Agentic-PRs as our main study scope about 5,000 samples, depending on available compute. To ensure representativeness, the subset will be stratified by repository and domain frequency, with roughly balanced positive and negative cases. Whenever possible, PRs will be paired within the same repository but with different acceptance outcomes to support comparative analysis. We will then use DeepSeek-V3 for large-scale labeling and cross-validate with GPT-5 and Claude 4.5 Sonnet to estimate model bias and consistency. This step can answer RQ2 and is expected to be completed by **Nov 9th** as well.
- Thirdly, we will analyze the correlation between PR features and outcomes, such as acceptance rate, review latency, comment density. A mixed-effects regression model will be used to quantify the effects of domain and agent identity on PR success. This task addresses RQ3 and is expected to be done by **Nov 23th**.

The final deliverables include:

- A labeled dataset of 5,000 Agentic-PRs with domain annotations;
- Domain-level statistical results on PR acceptance and review behavior;
- Reproducible labeling, evaluation, and analysis scripts for future research.

6 Threats and Mitigation

- **Sample Bias:** The selected subset of PRs may overrepresent certain repositories or domains. We will perform stratified sampling to balance domain frequency and repository activity.
- **Manually Labelled Data:** Human labels may vary due to subjective interpretation. Peer review and consensus checking will be applied to improve reliability.
- **LLM-Assisted Labelling:** Model-based labeling may reflect training-data bias. Cross-validation with multiple models (such as DeepSeek-V3, GPT-5, Claude 4.5) will be used to estimate bias and stability.

References

[1] 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.
[2] Claude Code. Claude code | claude, 2025.
[3] Cursor. Cursor - the ai code editor, 2024.
[4] GitHub. Github copilot · your ai pair programmer, 2025.
[5] Hao Li. Aidev, 2025.
[6] Chunhua Liu, Hong Yi Lin, and Patanamon Thongtanunam. Too noisy to learn: Enhancing data quality for code review comment generation, 2025.
[7] 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, 2025.
[8] 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.

A Appendix

Table 1: Prompt: Domain Classification

You are a senior software architect experienced across kernel, cloud-native, data, ML, and front-end stacks. Your job is to read the supplied **code diff** (and, if present, the related issue text) and decide which *technical domain(s)* the change touches.

Domain Tag Vocabulary
Use any tag from the list below **or create one that follows Rule 2**.

Core tags

`dtype, io, regex, concurrency, security, numerical, ml, graph, parsing, file-system, serialization, memory, performance, networking, database, cli, logging, config, datetime, unicode, path, container, build, packaging, versioning, gpu, distributed, caching, algorithm, recursion, state-machine, auth, crypto, api, schema, SQL, noSQL, docstring, l10n, i18n, accessibility, visualization, instrumentation, telemetry, overflow, underflow, NaN, encoding, compression, rand, seed, os-compat, symlink, exception, test-infra`

Extended tags

`admission-control, audit, batch, cloud, code-generation, conformance, controller, custom-resource, dns, docker, extensibility, federation, ha, hw-accel, images-registry, ingress, ipv6, network-policy, node-lifecycle, observability, provider, release-eng, reliability, secret, swagger, usability, io-avro, io-cloud, io-csv, io-database, io-delta, io-iceberg, io-json, io-parquet, interchange, interop, grpc, rest, ci, cd, devops, rollback, scheduler, metrics, tracing, feature-flag, a11y-aria, spark-sql, spark-streaming, spark-mllib, spark-graphx, comp:core, comp:data, comp:eager, comp:api, comp:dist-strat, comp:gpu, comp:gpu:tensorrt, comp:tpu, comp:xla, comp:keras, comp:grappler, comp:optimizer, module:autograd, module:nn, module:jit, module:onnx, module:cuda, module:quantization, module:distributed, module:fp16, module:dataloader, module:hub, module:lite, component:dom, component:concurrent-features, component:fast-refresh, component:devtools, component:build-infrastructure, browser:ie, browser:safari`

Rule 2 - Custom Tags If no existing tag fits, you may invent **one custom tag**. Custom tags must start with `X_`, contain 3 English words, and must not conflict semantically with the vocabulary above.

Output Guidelines

1. Return **1–3 tags** (comma-separated, no spaces, no quotes).
2. Choose the *most specific* tags; if both a parent and child fit, choose the child.
3. If multiple domains apply, list up to three in relevance order.
4. Provide **no explanations**.

Code Changes
`{code_changes}`
Issue Body
`{issue_body}`