

On Coding Agent Issue Localization Accuracy – An Exploratory Study

Antonino Coppola 

University of Southern Denmark, Vejle, Denmark

Matteo Esposito 

University of Southern Denmark, Vejle, Denmark

University of Oulu, Finland

Rick Kazman 

University of Hawaii, USA

Valentina Lenarduzzi 

University of Southern Denmark, Vejle, Denmark

Abstract

Context. Generative AI coding agents are increasingly used to automate software maintenance and issue resolution. However, current evaluations mainly focus on test-passing behavior and provide limited insights into whether agents modify the same software entities selected by developers.

Aim. This study investigates the accuracy of issue localization performed by agents across different software granularities.

Method. We analyzed 2,441 issue-fixing commits from 10 large-scale Java projects and evaluated three agents, each combining the OpenCode harness with a different open-weight LLM. We compared agent-modified entities against human-implemented fixes at the package, class, and method levels using Accuracy, Precision, Recall, F1-score, and MCC.

Results. Agents partially identified the software entities requiring modification, but localization performance strongly depended on software granularity. Agents achieved the strongest results at the package level, while performance progressively degraded at the class and method levels. GLM-5 consistently achieved the strongest localization performance, although practical differences among agents remained limited. Our findings showed a limitation of agents implementing functionally plausible yet structurally different packages, class or methods leading to consistently negative MCC values worsening with finer granularity.

Conclusion. Current agents can often identify the general architectural region affected by an issue, but still struggle to precisely localize fine-grained implementation points. These findings highlight both the potential and the limitations of agents for issue localization and motivate larger-scale investigations on localization behavior, architectural impact, and long-term maintainability implications.

2012 ACM Subject Classification Software and its engineering → Software defect analysis; Software and its engineering → Software architectures; Computing methodologies → Simulation types and techniques

Keywords and phrases AI, Coding Agents, Empirical Study

Digital Object Identifier 10.4230/LIPIcs.ESEM.2026.59

Category Emerging Results, Vision & Reflection Track Paper

1 Introduction

The rapid adoption of Large Language Model (LLM)-based generative AI agents is reshaping how developers write, modify, and maintain software. These tools can already generate syntactically correct and functionally plausible code for tasks such as bug fixing, feature



© Antonino Coppola, Matteo Esposito, Rick Kazman, and Valentina Lenarduzzi;
licensed under Creative Commons License CC-BY 4.0

20th International Symposium on Empirical Software Engineering and Measurement (ESEM 2026).

Editors: Robert Feldt, Maria Paasivaara, Daniel Mendez, Stefan Wagner, and Marvin Muñoz Barón;

Article No. 59; pp. 59:1–59:13



Leibniz International Proceedings in Informatics

LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

implementation, and refactoring. They are becoming part of everyday development workflows. However, despite the short-term productivity gains that they bring, their long-term impact on software quality and maintainability remains largely unexplored [1, 10].

A central concern is whether agents act *where* a human developer acted; that is, whether they modify the same parts of a codebase that a maintainer with deep project knowledge would have done. Recent benchmarks such as SWE-Bench [13] have shown that agents such as SWE-agent [32], OpenDevin [27], and AutoCodeRover [34] can resolve a substantial fraction of real GitHub issues, as measured by hidden test suites. However, test-based evaluation alone cannot distinguish *correct localization* (editing the right files, classes, and methods) from *incidental coverage*, where a patch satisfies the tests while modifying entities that the human developer would never have touched. This ambiguity has been documented in the Automated Program Repair (APR) literature as the *plausibility vs. correctness* distinction [19], but it remains largely unexplored for modern LLM-based agents.

To address these gaps, we designed and conducted an exploratory study on agent-based issue localization accuracy. We reconstructed historical issue-fixing contexts, reverted repositories to pre-fix states, generated agent-based fixes, and evaluated them against the human ground truth. We evaluated three OpenCode-based agent configurations using different underlying LLMs. We refer to such configurations as “agents” for the remainder of the paper.

Our study provides the following contributions:

- A multi-agent evaluation of how accurately agents identify the software entities requiring modification within a codebase.
- A fine-grained analysis of agent performance in localizing changes at the package, class, and method levels.
- A comparison between agents and human developers in terms of accuracy and precision of issue localization.

Paper Structure. Section 2 describes the empirical study design and analysis methodology. Section 3 presents our findings and Section 4 discusses them. Section 5 addresses the threats to validity of this work. Finally, Section 6 discusses the related works and Section 7 concludes the paper.

2 Empirical Study Design

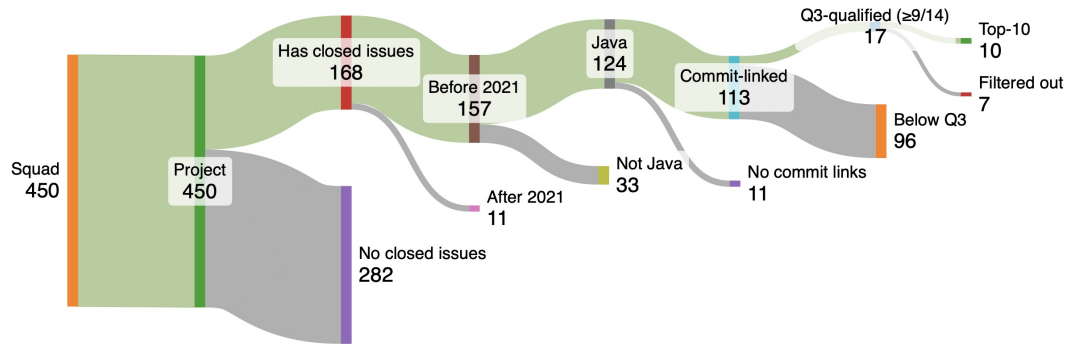
This section presents the empirical study design, including the study goal and research questions, the study context, the data collection methodology, and the data analysis approach [29].

2.1 Goal and Research Questions

We aim to analyze agent-generated solutions in issue-resolution tasks to evaluate their ability to accurately identify the software entities requiring modification, with respect to correctness and granularity (package, class, and method levels).

Therefore, we derived the following research question (RQ) from our goal:

RQ₁. Can agents accurately determine the software entities that must be modified to address a reported issue?



■ **Figure 1** Project filtering and selection process from the SQuaD dataset to the final study set.

Understanding where an issue should be resolved within a codebase is a central step in software maintenance and evolution [36]. With the increasing adoption of agents, an open question concerns their ability to localize the software entities requiring modification reliably. In this study, a **change** refers to the modification, addition, or removal of a software entity touched by a commit. We investigate whether agents can identify the correct **packages**, **classes**, and **methods** modified in the human-implemented reference fix. We operationalize localization performance using Accuracy, Precision, Recall, F₁-score, and Matthews Correlation Coefficient (MCC) computed by comparing agent-modified entities against the ground-truth human implementation. Since localization requires increasingly fine-grained contextual understanding, we expect localization performance to vary across software granularities. Therefore, we test the following hypotheses:

- $H_{0.1}$ There is no significant difference in localization performance among the evaluated agents.
- $H_{0.2}$ There is no significant difference in localization performance across package-, class-, and method-level software entities for agents.
- $H_{1.1}$ Localization performance significantly differs among the evaluated agents.
- $H_{1.2}$ Localization performance significantly differs across package-, class-, and method-level software entities for agents.

2.2 Study Context and LLM Selection

We based our **project selection** on SQuaD, introduced by Robredo et al. [21], a multi-dimensional, time-aware dataset of software quality metrics from 450 mature open-source projects. With over 700 metrics at the method, class, file, and project levels. Covering 63,586 releases, SQuaD also includes version control, issue tracking, vulnerability data (CVE/CWE), and process metrics for JIT defect prediction and over 628k issues. We applied a sequence of filters to obtain a focused subset of large-scale, mature Java projects as detailed in Section 2.3.

We implemented an **agent** using the OpenCode¹ framework to integrate LLMs into software engineering tasks. We consolidated three independent LLM ranking sources accessed in March 2026: Artificial Analysis², LLMstats³, and OpenCompass⁴, to identify the most

¹ <https://opencode.ai/>

² <https://artificialanalysis.ai>

³ <https://llm-stats.com>

⁴ <https://rank.opencompass.org.cn>

■ **Table 1** Overview of the evaluated LLMs.

Model	Parameters & Quantization	Developer	Details
GLM-5	744B, not quantized	Zhipu AI	Transformer-based MoE with DeepSeek Sparse Attention (DSA)
Kimi-K2.5	1T, not quantized	Moonshot AI	Transformer-based MoE with MLA attention and MoonViT 400M vision encoder
Qwen3.5	397B, not quantized	Alibaba	Hybrid architecture: Gated Delta Networks (linear attention) + Transformer MoE

performant models for code-related tasks across different evaluation metrics. Artificial Analysis employs Terminal Bench Hard and SciCode benchmarks, LLMstats leverages the Code-Arena benchmark, and OpenCompass utilizes the LiveCodeBench⁵ metric for assessment. Following consolidation, we computed the median ranking position across these sources and subsequently selected the top-performing models demonstrating consensus across multiple ranking systems. Based on this methodology, we identified three models for experimental evaluation: GLM-5⁶, Kimi-K2.5⁷ and Qwen3.5⁸ (Table 1).

2.3 Data Collection

We divided our data collection workflow into three processing steps, as follows.

(1) Project and Issue Collection. The final study set was constructed using SQuAD [21] as the initial reference corpus, followed by a multi-stage filtering and sampling procedure as illustrated in Figure 1. At the project level, we restricted the corpus to Java repositories hosted on GitHub with closed issues linked to commits and tracked on JIRA. We retained only issues closed before January 1st, 2021. This cutoff predates the widespread availability of LLM-based coding assistants, ensuring that the human-implemented fixes used as ground truth reflect unassisted developer decisions rather than AI-suggested ones. We then ranked projects according to 14 SQuAD size-and-maturity metrics [21] covering repository scale, code volume, defect history, longevity, and tracker availability, and selected the top 10 projects ranking above the third quartile on at least 9 of the 14 dimensions. The selected projects are listed in Table 2. At the issue level, we applied quality filters according to previous works that investigated similar problems. In particular, we selected issues that were implemented at most across 3 fix commits [11] touching at most 10 Java files [17]. The number of surviving issues per project is reported in the *Issues* column of Table 2. Then, we deterministically extracted a statistically significant sample population of n_p with 95% confidence and 5% margin of error [5]. We chronologically sorted the surviving issues by first fix-commit date, and we selected n_p issues at uniformly spaced indices across the temporal range. The per-project sample sizes are reported in the *Sample(n_p)* column of Table 2.

(2) Collecting Human-Implemented Issue-Fixing Commits. We linked each issue to its corresponding implementation commit(s), referred to as Human-Implemented Issue-Fixing Commits (HIIFC). Since issue-to-commit linkage is inherently noisy, we adopted a best-effort

⁵ <https://livecodebench.github.io>

⁶ <https://ollama.com/library/glm-5>

⁷ <https://ollama.com/library/kimi-k2.5>

⁸ <https://ollama.com/library/qwen3.5:397b-cloud>

■ **Table 2** Overview of the selected projects, issue counts, sampled issues, and completed executions.

Project	Domain	Issues	Sample (n_p)	Completed
Camel	Integration framework	8,103	368	329
Cloudstack	Cloud platform	3	3	3
Flink	Stream processing	5,728	361	335
Hadoop	Batch processing	1,618	312	310
Hbase	Distributed DB	9,407	370	363
Hive	Data warehouse SQL	10,089	371	340
Ignite	In-memory database	3,559	348	326
Iotdb	Time-series DB	228	144	189
Netbeans	IDE platform	558	229	245
Cxf	Web services framework	1	1	1
Total		39,294	2,507	2,441

approach based on the regular expression matching used by Robredo et al. [20], specifically the JIRA pattern `[A-Z]{2,}-\d+`. We acknowledge that this approach may miss some issue-fixing commits due to heterogeneous commit practices, and we discuss this as a threat to validity. For each HIFC, we extracted the commit metadata, including message, diff, touched files, author, timestamp, and parent commit. The parent of the earliest HIFC, which we refer to as the Pre-Human-Initial-Fix-Commit (PHIFC), represents the exact context available to developers before the fix and serves as the reference state for our analysis. To extract the ground truth, we check out the repository at PHIFC in a detached worktree and apply the cumulative diff between PHIFC and HIFC restricted to `.java` files. We parse each pre-fix file with `tree-sitter`⁹ and identify the Abstract Syntax Tree (AST) nodes whose source range overlaps the changed lines, projecting them to three granularities: package, class, and method.

(3) Agent-Based Re-Implementation. We then instantiate a fresh `git worktree` at PHIFC for the agent and remove references to the post-PHIFC state to prevent future-state leakage. We invoke the agent through the OpenCode harness with a model-agnostic prompt that provides the issue description and the corresponding codebase snapshot and instructs the agent to produce a fix. The resulting unified diff constitutes the Agent-Implemented Issue-Fixing Commit (AIFC), from which we extract entities using the same `tree-sitter` pipeline employed for ground truth. We evaluate three agents, each instantiated by pairing the OpenCode harness with one of the open-weight LLMs listed in Section 2.2, and accessed through Ollama Cloud¹⁰. All inference is performed remotely on Ollama’s infrastructure. We did not provide any information from the human implementation, such as diffs, commit messages, future code states, nor access to tools, i.e., `git`, that would reveal the commit history to the agent.

Validity and Timeout Handling. A non-trivial fraction of agent runs failed in the initial execution due to a 600-second timeout. Rather than re-executing all timed-out runs, we selectively re-ran issues from projects exhibiting a high number of timeout failures using a

⁹ <https://tree-sitter.github.io/tree-sitter/>

¹⁰ <https://ollama.com>

1,200-second timeout, to verify whether the time bound was a significant constraint. The number of issues recovered through extended execution varies substantially across models: Qwen3.5 recovered 139 issues, GLM-5 recovered 18, and Kimi-K2.5 recovered only one. After the retry, we classify a run as valid only if it completed with exit code 0 and did not time out. Out of 2,506 total runs per model, Qwen3.5 failed on 144 issues (5.74%), GLM-5 on 44 issues (1.75%), and Kimi-K2.5 on 59 issues (2.35%): verbose-reasoning models are markedly more affected by the timeout budget. To ensure fair cross-model comparison, we restrict the analysis to the intersection of valid issues, meaning the 2,441 issues for which all three models produced a valid run, for a total of 7,323 executions. Since each execution is evaluated at three granularity levels, this yields 21,969 granularity-level observations used in the analysis.

2.4 Data Analysis

This section outlines the data analysis procedure adopted to answer the research questions and evaluate the associated hypotheses. To identify the appropriate statistical techniques, we first assessed the distribution of the collected data [9] through the following hypotheses:

$H_{0\mathcal{N}}$ *The collected data is normally distributed.*

$H_{1\mathcal{N}}$ *The collected data is not normally distributed.*

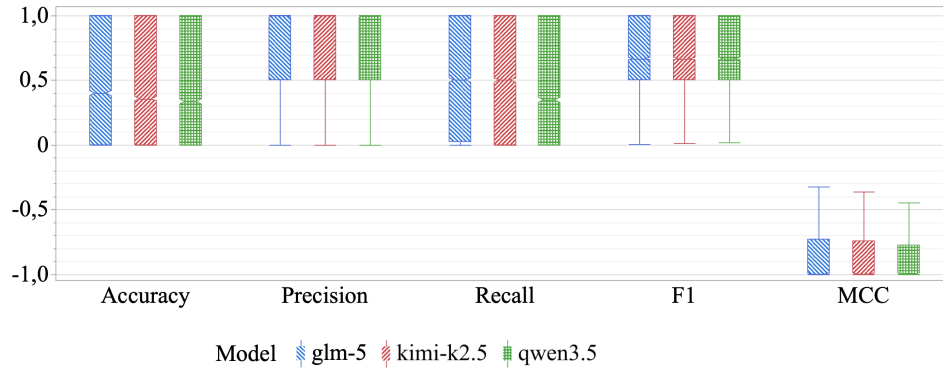
We tested $H_{0\mathcal{N}}$ using the Anderson-Darling test [2], which is suitable for large datasets and sensitive to departures from normality [25]. Across all RQs, we report the statistical test, p -value, effect size, and practical interpretation of the results. We adopted a significance threshold of $\alpha = 0.05$, applying Bonferroni correction [8, 29] and additionally reporting Benjamini-Hochberg-adjusted (BH) p -values [4] as sensitivity analysis.

To test H_{01} and H_{02} , we analyzed localization performance across package-, class-, and method-level entities using Accuracy, Precision, Recall, F1-score, and MCC. Since the data followed a repeated-measures design and normality was rejected in all cases, we used the Friedman test followed by post hoc Wilcoxon signed-rank tests with multiple-comparison correction. We report Kendall’s W and rank-biserial correlation (r_{rb}) [7] as effect sizes.

3 Results

This section presents the findings of our investigation and addresses our RQ. We computed all results on the intersection of issues for which all three agents produced a valid run, resulting in 2,441 issues across 10 projects.

Figure 2 shows the distributions of Accuracy, Precision, Recall, F1-score, and MCC across the evaluated agents. GLM-5 achieved the strongest localization performance across most metrics, particularly for Recall and MCC, while Qwen3.5 and Kimi-K2.5 showed lower and more variable performance. Across all agents, Precision remained comparatively high whereas Recall remained substantially lower, indicating that agents often identified relevant entities correctly but failed to capture the full set of entities modified by developers. Moreover, MCC values remained consistently negative, suggesting limited structural agreement with the human-implemented fixes despite partial overlap in modified entities. Since the Anderson-Darling test rejected normality in all cases, we applied the Friedman test followed by Wilcoxon signed-rank post-hoc analyses. The omnibus analysis revealed statistically significant differences among agents for all localization metrics. However, all Kendall’s W effect sizes remained negligible, indicating limited practical differences among models. Recall produced the strongest separation among agents, whereas F1-score showed no significant pairwise differences despite the significant omnibus result (Table 3). Therefore, the results



■ **Figure 2** Distribution of localization metrics across the evaluated agents, aggregated over all granularity levels (package, class, and method).

■ **Table 3** Statistical comparison of localization performance across agents, aggregated over all granularity levels (package, class, and method).

Metric	Best Group	Mean	Significant	Posthoc Summary
Accuracy	GLM-5	0.473	✓	2/3 both, 0/3 borderline, 1/3 ns
Precision	GLM-5	0.756	✓	2/3 both, 0/3 borderline, 1/3 ns
Recall	GLM-5	0.492	✓	3/3 both, 0/3 borderline, 0/3 ns
F1	GLM-5	0.714	✓	0/3 both, 0/3 borderline, 3/3 ns
MCC	GLM-5	-0.840	✓	2/3 both, 0/3 borderline, 1/3 ns

reject $H_{0.1}$: agents differ in localization performance, although the practical differences among models remain limited. Table 3 aggregates results across all granularity levels, a per-granularity breakdown is reported in Table 4, the two should be read together rather than compared directly.

Figure 3 summarizes localization performance across package-, class-, and method-level entities. Localization quality degrades as granularity becomes finer: Accuracy, Recall, and F1-score consistently decrease from packages to classes to methods. At the package level, agents achieved their strongest overall performance, whereas at the method level Accuracy and Recall reached their lowest values, indicating difficulties in pinpointing the exact implementation points modified by developers.

GLM-5 achieved the best mean performance across most metrics and granularities; exceptions are package-level Precision and class-level MCC, where Kimi-K2.5 ranked highest (Table 4). Nevertheless, the practical differences among agents remained limited at every granularity, as reflected by negligible Kendall’s W values despite several statistically significant comparisons. Method-level MCC showed significant differences among agents, suggesting that fine-grained localization amplifies behavioral differences between models.

Overall, the results reject $H_{0.2}$: **localization quality degrades as granularity becomes finer**. Agents reliably localized coarse-grained entities, whereas fine-grained method-level localization remained substantially more challenging. The architectural region of the issue is often correctly identified, but the exact implementation points modified by developers are not.

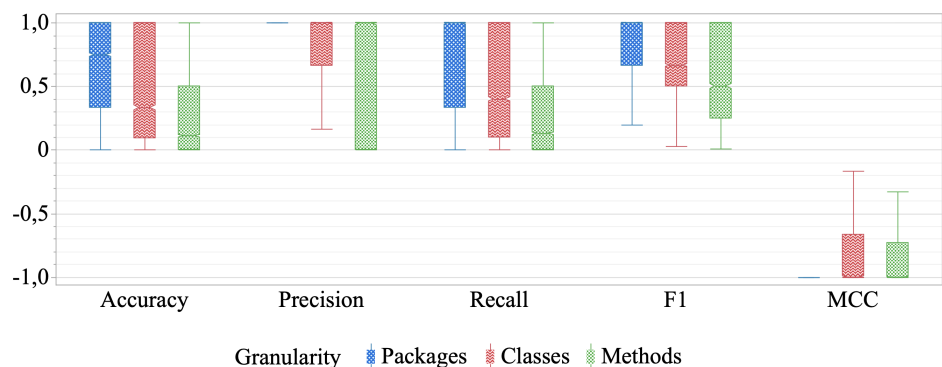


Figure 3 Distribution of localization metrics across package-, class-, and method-level entities.

Table 4 Statistical comparison of localization performance across software granularities.

Granularity	Metric	Best Group	Best Mean	Significant	Posthoc Summary
packages	Accuracy	GLM-5	0.646	✓	2/3 both, 0/3 borderline, 1/3 ns
	Precision	Kimi-K2.5	0.877	✗	No post-hoc (omnibus ns)
	Recall	GLM-5	0.661	✓	2/3 both, 0/3 borderline, 1/3 ns
	F1	GLM-5	0.833	✗	No post-hoc (omnibus ns)
	MCC	GLM-5	-0.906	✗	No post-hoc (omnibus ns)
classes	Accuracy	GLM-5	0.459	✓	2/3 both, 1/3 borderline, 0/3 ns
	Precision	GLM-5	0.779	✓	1/3 both, 1/3 borderline, 1/3 ns
	Recall	GLM-5	0.479	✓	3/3 both, 0/3 borderline, 0/3 ns
	F1	GLM-5	0.686	✓	2/3 both, 0/3 borderline, 1/3 ns
	MCC	Kimi-K2.5	-0.858	✗	No post-hoc (omnibus ns)
methods	Accuracy	GLM-5	0.296	✓	2/3 both, 0/3 borderline, 1/3 ns
	Precision	GLM-5	0.597	✓	1/3 both, 0/3 borderline, 2/3 ns
	Recall	GLM-5	0.316	✓	3/3 both, 0/3 borderline, 0/3 ns
	F1	GLM-5	0.562	✓	2/3 both, 0/3 borderline, 1/3 ns
	MCC	GLM-5	-0.861	✓	2/3 both, 0/3 borderline, 1/3 ns

4 Discussion

As expected, agents localized coarse-grained software entities substantially better than fine-grained ones: the larger the target, the easier it is to hit. Performance progressively degraded from packages to classes and methods, indicating that current agents can often identify the general architectural region affected by an issue, but still struggle to precisely pinpoint the exact implementation locations. In particular, the **strong reduction in Recall at the method level** suggests that agents frequently miss relevant implementation points even when they partially identify the correct area of the codebase.

Despite the reduction in Recall, **Precision remained comparatively high across all evaluated models and granularities**. Agents rarely modified completely unrelated entities and instead tended to modify only a subset of the entities touched by developers. This conservative behavior may reduce invasive or destabilizing edits, but it also increases the likelihood of incomplete fixes, especially when issues require coordinated modifications across multiple methods or classes.

The **consistently negative MCC values** indicate limited structural agreement between agent- and human-modified entities. In several cases, agents modified entities different from those touched by developers, yet may still have implemented semantically equivalent behavior, a possibility that overlap-based metrics alone cannot confirm or rule out. This raises the question of whether such structurally divergent solutions are genuinely incorrect or merely alternative implementations of the same fix. To quantify the potential extent of this effect, we identified the divergent cases, i.e., observations in which the agent both modified entities not touched by the developer and missed entities the developer modified. Such cases account for 25.07% of all 21,969 observations, and their proportion grows with finer granularity: 11.73% at the package level, 23.42% at the class level, and 40.05% at the method level (7,323 observations each), with consistent proportions across models (24.65% - 25.70%). We discuss the implications of this measurement limitation for the validity of our findings in Section 5.

The negligible effect sizes across models suggest that localization limitations currently represent a broader characteristic of agents rather than a weakness of a specific model family. Although GLM-5 achieved the strongest overall performance, the practical differences among agents remained limited despite large differences in model scale. These findings raise an important question: do substantially larger models actually improve issue localization quality? Current results suggest that model scaling alone may not be sufficient, and that future agents will likely require stronger repository-aware reasoning mechanisms, including dependency analysis, structural navigation, execution-context understanding, and architectural awareness. From a practical standpoint, these results suggest that coding agents can already provide useful support in the early triaging phase of issue resolution, helping maintainers narrow down the relevant architectural area. However, they should not be relied upon for precise method-level localization without human verification, as the high rate of divergent cases at finer granularities indicates a substantial risk of incomplete or misplaced fixes.

Future research should investigate the relationship between localization quality and long-term software quality indicators, including maintainability, architectural degradation, and technical debt accumulation. Moreover, future evaluation frameworks should move beyond **strict entity-overlap metrics** and incorporate semantic and behavioral equivalence between human and agent implementations. Finally, larger-scale and longitudinal studies are needed to better understand the characteristics and long-term impact of agent-based issue localization and re-implementation.

5 Threats to Validity

This section discusses the main threats to validity of our study following established guidelines [29].

Construct Validity. We operationalize issue localization through Accuracy, Precision, Recall, F_1 -score, and MCC computed against human-implemented fixes. As discussed in Section 4, our overlap-based metrics treat any divergent case as incorrect, even when the agent may have produced a semantically equivalent fix through different entities. These metrics therefore cannot recognize functionally valid but structurally divergent solutions, and our findings should be interpreted as a lower bound on agent localization performance. Consistent with the scope of an Emerging Results paper, a systematic analysis of functional equivalence on these cases is left to future work. Additionally, our results should be read as differences across LLMs within a fixed orchestration environment rather than as a general comparison of agent frameworks. Finally, fix commits may contain spurious changes unrelated to the reported issue, such as unrelated refactorings or formatting updates, which could inflate the ground-truth entity set and affect the overlap-based metrics.

Internal Validity. We link issues to commits through regular-expression matching, which may introduce noise due to inconsistent commit practices. To mitigate this limitation, we adopted established matching patterns and selected issues with clear traceability. Agent-generated outputs may also vary because of prompt design, issue complexity, and model behavior. So, we standardized prompts and execution settings across all evaluated agents. Furthermore, LLM outputs are inherently non-deterministic, each (agent, issue) pair was executed once due to the computational cost of 7,323 total runs, which may not fully capture the variance in agent localization behavior. We also did not characterize the quality of the issue descriptions provided to the agents. JIRA reports vary considerably in detail, from one-line titles to structured descriptions with stack traces and reproduction steps, and this variation likely interacts with localization difficulty. Controlling for issue-report quality is a natural extension of this work.

External Validity. All three agents share the same OpenCode orchestration layer, and our results therefore characterize how different LLMs behave within a fixed harness rather than how agents behave in general. Harness design shapes repository exploration, context construction, and editing strategy, so localization behavior may differ under alternative frameworks. Our study focuses on open-source Java projects and may not generalize to industrial systems, other programming languages, or alternative development ecosystems. Finally, the pre-2021 cutoff guarantees an AI-free ground truth but does not exclude these issues from the models’ training corpora, agents may therefore have encountered some of these fixes during training.

Conclusion Validity. We mitigated statistical conclusion threats by testing data normality, applying non-parametric statistical tests, reporting effect sizes, and adopting multiple-comparison correction procedures. Nevertheless, several statistically significant results showed negligible effect sizes, indicating limited practical differences among models.

6 Related Work

Classical bug localization research relies on developer fix commits as ground truth and ranks source files or methods according to their similarity to bug reports [35, 22, 30]. Later studies introduced deep learning approaches to improve localization performance [14, 31]. However, these approaches treat localization as a static ranking problem and do not capture the exploratory behavior of agents, which actively navigate and modify the codebase before producing a solution.

Recent work has introduced agentic approaches for automatically identifying the code entities requiring modification. ORCALOCA [33], COSIL [12], and COGNIGENT [23] rely on iterative exploration strategies based on code graphs, reasoning, and multi-agent workflows to localize buggy entities. TOOLTRAIN [15] and LLMLOC [16] instead improve localization through enhanced retrieval and training mechanisms. Batole et al. [3] focus on identifying refactoring targets through static-analysis summaries. While these studies mainly propose techniques to improve localization, they do not empirically analyze whether agents modify the same software entities selected by developers across different granularities. In parallel, empirical studies have analyzed how AI-generated code differs from human-written code in terms of defects, vulnerabilities, complexity, readability, coding style, and structural properties [6, 28, 26, 18]. These works focus on the characteristics of the generated code after the modification has been applied. In contrast, our work investigates whether agents act in the *right place* to begin with by comparing agent-modified entities against the human-implemented fix at the package, class, and method levels.

Our work is also complementary to Automated Program Repair studies on plausible versus correct patches [24]. While APR research focuses on whether generated patches satisfy tests despite diverging from human fixes, we focus on the localization perspective, analyzing where agents apply modifications and how closely these locations align with developer-selected entities.

7 Conclusion

Agents can partially localize the software entities that require modification, but their performance strongly depends on software granularity. Agents reliably identified coarse-grained package-level entities; however accuracy progressively degrades as we move to the class and method levels, revealing the difficulty in pinpointing the exact areas that developers modified. Although GLM-5 achieved the strongest localization performance across most metrics and granularities, the practical differences among models are limited. Our results also show that agents frequently produce structurally divergent solutions whose functional equivalence to the human fix cannot be established through overlap-based metrics, which may therefore underestimate the practical usefulness of some generated solutions. Overall, these findings highlight both the potential and the current limitations of agents for issue localization and re-implementation tasks. Future large-scale studies should further investigate localization behavior, architectural impact, and the long-term characteristics of agent-generated modifications across diverse software systems and development scenarios.

Data Availability Statement

We provide all raw data, scripts, model prompts, and analysis configurations in a FAIR¹¹ replication package on Zenodo¹².

Declaration on the use of generative AI

The authors used ChatGPT for suggestions to improve textual clarity. All research design, data analysis, interpretation, and manuscript writing was conducted by the authors.

References

- 1 Bassam Adnan, Matteo Esposito, Davide Taibi, and Karthik Vaidhyanathan. Can ai agents generate microservices? how far are we? In *International Conference on Software Architecture (ICSA)*. IEEE, 2026. doi:10.48550/arXiv.2603.09004.
- 2 T. W. Anderson and D. A. Darling. Asymptotic theory of certain "goodness of fit" criteria based on stochastic processes. *The Annals of Mathematical Statistics*, 23(2):193–212, 1952.
- 3 Fraol Batole, David O'Brien, Tien N. Nguyen, Robert Dyer, and Hriday 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. doi:10.1109/ICSE55347.2025.00100.
- 4 Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society, Series B*, 57(1):289–300, 1995. doi:10.1111/j.2517-6161.1995.tb02031.x.

¹¹<https://www.go-fair.org/fair-principles/>

¹²<https://doi.org/10.5281/zenodo.20327573>

- 5 William G. Cochran. *Sampling Techniques, 3rd Edition*. John Wiley, New York, 3 edition, 1977.
- 6 Domenico Cotroneo, Cristina Improta, and Pietro Liguori. Human-written vs. ai-generated code: A large-scale study of defects, vulnerabilities, and complexity. In *International Symposium on Software Reliability Engineering (ISSRE)*, pages 252–263, 2025. doi:10.1109/ISSRE66568.2025.00035.
- 7 Edward E. Cureton. Rank-biserial correlation. *Psychometrika*, 21(3):287–290, 1956. doi:10.1007/BF02289138.
- 8 O. J. Dunn. Multiple comparisons among means. In *Journal of the American Statistical Association*, pages 52–64, 1961. doi:10.1080/01621459.1961.10482090.
- 9 Matteo Esposito, Mikel Robredo, Murali Sridharan, Guilherme Horta Travassos, Rafael Peñaloza, and Valentina Lenarduzzi. A critical reflection on the state of data analysis in empirical software engineering. *ACM Trans. Softw. Eng. Methodol.*, February 2026. doi:10.1145/3799715.
- 10 Hamed Fawareh and Fahad Muzil Alanazi. The significance of ai technologies to enhance productivity, efficiency, and software quality within the software development process. In *Artificial Intelligence for Sustainable Innovation Management and Risk Management: A Systems (and Network) Perspective*, pages 2435–2451. Springer Nature, 2026. doi:10.1007/978-3-031-95310-1_177.
- 11 Kim Herzig and Andreas Zeller. The impact of tangled code changes. In *Working Conference on Mining Software Repositories (MSR)*, pages 121–130, 2013. doi:10.1109/MSR.2013.6624018.
- 12 Zhonghao Jiang, Xiaoxue Ren, Meng Yan, Wei Jiang, Yong Li, and Zhongxin Liu. Issue localization via llm-driven iterative code graph searching. In *40th IEEE/ACM International Conference on Automated Software Engineering, ASE 2025, Seoul, Korea, Republic of, November 16-20, 2025*, pages 3034–3045. IEEE, 2025. doi:10.1109/ASE63991.2025.00249.
- 13 Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024. doi:10.48550/arXiv.2310.06770.
- 14 An Ngoc Lam, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N. Nguyen. Bug localization with combination of deep learning and information retrieval. In *International Conference on Program Comprehension (ICPC)*, pages 218–229, 2017. doi:10.1109/ICPC.2017.24.
- 15 Zexiong Ma, Chao Peng, Qunhong Zeng, Pengfei Gao, Yanzhen Zou, and Bing Xie. Tool-integrated reinforcement learning for repo deep search, 2025. doi:10.48550/arXiv.2508.03012.
- 16 Gyumin Nam and Geunseok Yang. Llmloc: A structure-aware retrieval system for zero-shot bug localization. *Electronics*, 14(21), 2025. doi:10.3390/electronics14214343.
- 17 Feifei Niu, Christoph Mayr-Dorn, Wesley K. G. Assunção, LiGuo Huang, Jidong Ge, Bin Luo, and Alexander Egyed. The ablots approach for bug localization: is it replicable and generalizable? In *International Conference on Mining Software Repositories (MSR)*, pages 576–587, 2023. doi:10.1109/MSR59073.2023.00083.
- 18 Abhi Patel, Kazi Zakia Sultana, and Bharath K. Samanthula. A comparative analysis between ai generated code and human written code: A preliminary study. In *International Conference on Big Data (BigData)*, pages 7521–7529, 2024. doi:10.1109/BigData62323.2024.10825958.
- 19 Zichao Qi, Fan Long, Sara Achour, and Martin Rinard. An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, page 24–36, 2015. doi:10.1145/2771783.2771791.
- 20 Mikel Robredo, Matteo Esposito, Fabio Palomba, Rafael Peñaloza, and Valentina Lenarduzzi. Analyzing the ripple effects of refactoring. *Empir. Softw. Eng.*, 31(5):134, 2026. doi:10.1007/s10664-025-10801-3.
- 21 Mikel Robredo, Matteo Esposito, Davide Taibi, Rafael Peñaloza, and Valentina Lenarduzzi. Squad: The software quality dataset. In *Proceedings of the 23rd International Conference on Mining Software Repositories*, pages 615–619. Association for Computing Machinery, 2026. doi:10.1145/3793302.3793312.

- 22 Ripon K. Saha, Matthew Lease, Sarfraz Khurshid, and Dewayne E. Perry. Improving bug localization using structured information retrieval. In *International Conference on Automated Software Engineering (ASE)*, pages 345–355, 2013. doi:10.1109/ASE.2013.6693093.
- 23 Asif Mohammed Samir and Mohammad Masudur Rahman. Improved Bug Localization with AI Agents Leveraging Hypothesis and Dynamic Cognition, February 2026. arXiv:2601.12522 [cs.SE]. doi:10.48550/arXiv.2601.12522.
- 24 Edward K. Smith, Earl T. Barr, Claire Le Goues, and Yuriy Brun. Is the cure worse than the disease? overfitting in automated program repair. In Elisabetta Di Nitto, Mark Harman, and Patrick Heymans, editors, *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, ESEC/FSE 2015, pages 532–543. ACM, 2015. doi:10.1145/2786805.2786825.
- 25 M. A. Stephens. Edf statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 69(347):730–737, 1974. doi:10.2307/2286009.
- 26 Wannita Takerngsaksiri, Chakkrit Tantithamthavorn, Michael Fu, Jirat Pasuksmit, Kun Chen, and Ming Wu. Code readability in the age of large language models: An industrial case study from atlassian. In *IEEE International Conference on Software Maintenance and Evolution, ICSME 2025, Auckland, New Zealand, September 7-12, 2025*, pages 732–742. IEEE, 2025. doi:10.1109/ICSME64153.2025.00073.
- 27 Xingyao Wang, Boxuan Li, Yufan Song, et al. Openhands: An open platform for ai software developers as generalist agents, 2025. doi:10.48550/arXiv.2407.16741.
- 28 Yanlin Wang, Tianyue Jiang, Mingwei Liu, Jiachi Chen, Mingzhi Mao, Xilin Liu, Yuchi Ma, and Zibin Zheng. Beyond functional correctness: Investigating coding style inconsistencies in large language models. *Proc. ACM Softw. Eng.*, 2(FSE), June 2025. doi:10.1145/3715749.
- 29 Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering, Second Edition*. Springer, 2024. doi:10.1007/978-3-662-69306-3.
- 30 W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Trans. Softw. Eng.*, 42(8):707–740, August 2016. doi:10.1109/TSE.2016.2521368.
- 31 Yan Xiao, Jacky Keung, Kwabena Ebo Bennin, and Qing Mi. Improving bug localization with word embedding and enhanced convolutional neural networks. *Inf. Softw. Technol.*, 105:17–29, 2019. doi:10.1016/j.infsof.2018.08.002.
- 32 John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering, 2024. doi:10.48550/arXiv.2405.15793.
- 33 Zhongming Yu, Hejia Zhang, Yujie Zhao, Hanxian Huang, Matrix Yao, Ke Ding, and Jishen Zhao. OrcaLoca: An LLM Agent Framework for Software Issue Localization, October 2025. arXiv:2502.00350 [cs.SE]. doi:10.48550/arXiv.2502.00350.
- 34 Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement, 2024. doi:10.48550/arXiv.2404.05427.
- 35 Jian Zhou, Hongyu Zhang, and David Lo. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In *International Conference on Software Engineering (ICSE)*, pages 14–24, 2012. doi:10.1109/ICSE.2012.6227210.
- 36 Jonida Çarka, Matteo Esposito, and Davide Falessi. On effort-aware metrics for defect prediction. *Empir. Softw. Eng.*, 27(6):152, 2022. doi:10.1007/s10664-022-10186-7.