

How Do Agents Perform Code Optimization? An Empirical Study

Huiyun Peng
Purdue University
West Lafayette, Indiana, USA

Kelechi G. Kalu
Purdue University
West Lafayette, Indiana, USA

Antonio Zhong
Purdue University
West Lafayette, Indiana, USA

Ricardo Andrés Calvo Méndez
Purdue University
West Lafayette, Indiana, USA

James C. Davis
Purdue University
West Lafayette, Indiana, USA

Abstract

Performance optimization is a critical yet challenging aspect of software development, often requiring a deep understanding of system behavior, algorithmic tradeoffs, and careful code modifications. Although recent advances in AI coding agents have accelerated code generation and bug fixing, little is known about how these agents perform on real-world performance optimization tasks.

We present the first empirical study comparing agent- and human-authored performance optimization commits, analyzing 324 agent-generated and 83 human-authored PRs from the AIDev dataset across adoption, maintainability, optimization patterns, and validation practices. We find that AI-authored performance PRs are less likely to include explicit performance validation than human-authored PRs (45.7% vs. 63.6%, $p = 0.007$). In addition, AI-authored PRs largely use the same optimization patterns as humans. We further discuss limitations and opportunities for advancing agentic code optimization.

CCS Concepts

- Software and its engineering → Code Optimization.

Keywords

Software Performance Optimization, AI Coding Agents

ACM Reference Format:

Huiyun Peng, Antonio Zhong, Ricardo Andrés Calvo Méndez, Kelechi G. Kalu, and James C. Davis. 2025. How Do Agents Perform Code Optimization? An Empirical Study. In *Proceedings of MSR '26: Proceedings of the 23rd International Conference on Mining Software Repositories (MSR 2026)*. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 Introduction

Software performance is central to the reliability [19], user experience [44], and energy efficiency [21] of modern systems. Despite decades of advances in compilers [27], algorithms [3], and hardware-aware tuning [7], performance engineering remains a labor-intensive task requiring specialized expertise and systematic profiling [4]. Large Language Models (LLMs) are increasingly used

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MSR 2026, Rio de Janeiro, Brazil

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

for software development tasks such as code synthesis, debugging, and refactoring [31], and have shown promise in improving non-functional properties like runtime and energy efficiency [13, 33].

Autonomous coding agents are increasingly contributing substantial numbers of PRs across open-source software, including performance-related changes [42]. Performance optimization, however, poses distinct challenges: performance-oriented PRs often have non-local effects, rely on implicit assumptions about workloads or hardware, and require strong empirical evidence (e.g., benchmarking or profiling) to justify their correctness and benefit [12]. As AI agents begin to author such changes at scale, it becomes critical to understand how performance improvements are carried out by humans and AI in real-world settings.

To address this gap, we conduct the first empirical study of performance-focused pull requests drawn from the AIDev [22] dataset, comparing agent and human-authored performance PRs via the SysLLMatic [34] catalog. We ask:

- **RQ1:** How do AI agents and humans differ in the optimization patterns they apply in performance-oriented patches?
- **RQ2:** How do AI agents and humans differ in their testing and validation of performance improvements?

We answer these questions using a subset of 407 performance-related PRs (4,954 commit details) from the AIDev dataset. We find that AI-authored performance PRs rely on optimization patterns similar to those used by humans, but offer weaker validation of performance impact and mostly rely on static reasoning rather than benchmarking. In addition, we contribute an extended performance optimization catalog, introducing 14 new patterns spanning two additional categories to better capture the diversity of real-world performance patches. This work advances our understanding of real-world performance optimization practices and highlights both the strengths and limitations of AI agents in this domain.

2 Background & Related Work

2.1 Performance Engineering

2.1.1 Optimization Patterns. Performance engineering spans the full software stack, from application logic to operating system and hardware behavior. At the source-code level, performance tuning involves revising algorithms and data structures, improving control flow and memory locality, and exploiting parallelism – often guided by a system-wide view of where time and resources are spent [16].

To answer RQ1, we require a concrete catalog of such optimization methods so we can consistently characterize the strategies used in performance-related PRs. SysLLMatic [34] provides this

foundation by introducing the state-of-the-art catalog of 43 optimization patterns across seven categories, and demonstrating how the catalog can be applied to guide optimizations on real-world benchmarks. We adopt this catalog as a shared vocabulary for describing performance improvements in our dataset.

2.1.2 Performance Validation. Optimizations require rigorous validation because performance is highly sensitive to measurement noise, workload choice, and experimental design; best practices therefore emphasize representative workloads, repeatable protocols, and statistically grounded comparisons [19].

To answer RQ2, we require a characterization of performance validation practices so we can systematically assess how performance claims are supported in perf PRs. We focus on three common forms of performance validation evidence used in software engineering practice. First, *Benchmark-Based Validation* relies on quantitative measurements (e.g., unit tests or microbenchmarks) to directly assess performance impact [18, 19]. Second, *Profiling-Based Validation* uses profiling artifacts such as hotspot traces, CPU samples, or flame graphs to localize performance costs [15]. Third, *Static-Reasoning-Based Validation* supports performance claims through algorithmic arguments without relying on runtime data, e.g., by identifying asymptotic inefficiencies [5, 30]. These three forms capture the primary ways developers justify performance improvements.

2.2 Related Work

Prior works have mined real-world patches to study performance optimization. RAPGen [10] mines performance-related PRs to build an optimization knowledge base for fixing inefficiencies, while Yi *et al.* [46] uses human-authored performance PRs as ground truth, prompt LLMs to optimize the original code, and compare the results against the developers' patches. Concurrently, recent work has begun empirically examining agent PRs in real-world repositories, comparing their characteristics and integration outcomes with human-authored PRs [20, 40, 42]. However, prior work has not compared agent- and human-authored performance pull requests or analyzed performance factors such as optimization strategies and validation practices. To fill this gap, we present the first empirical comparison of agent- and human-authored performance PRs, analyzing optimization patterns and validation practices.

3 Methodology

This section describes our data collection pipeline and analysis methods for performance-related PRs, as summarized in Figure 1.

3.1 Data Preparation

We conduct our study using AIDEV [22], a dataset of GitHub pull requests (PRs) authored by agents and humans with commit, diff, review, and repository-level metadata. We use two subsets: AIDEV-Pop (33,596 PRs; repositories with ≥ 100 stars), and HUMAN-PR (6,618 PRs; repositories with ≥ 500 stars). We focus on performance-related PRs (label: perf), yielding 340 agent PRs and 88 human PRs.

To enable code-level analysis, we require fine-grained diff data. Because HUMAN-PR omits commit-level diff data, we mine commit details from GitHub and reconstruct 7,874 commit-detail records for the 88 human perf PRs; combined with AIDEV-Pop, this yields 15,284 commit-detail records across the 428 perf PRs. We then

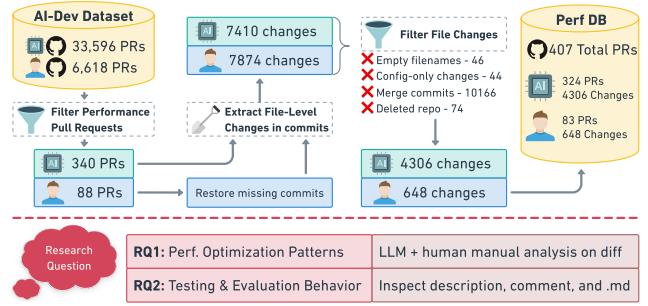


Figure 1: Methodology overview illustrating the data collection pipeline and how the resulting dataset supports answering RQ1 and RQ2.

apply quality filters, removing records with missing filenames (46), configuration-only changes (44), merge commits (10,166), and commits from deleted repositories (74), resulting in 4,954 valid commit-detail records corresponding to 407 perf PRs with analyzable patches. Coverage remains high, with valid commits in 324/340 AI-agent PRs (95.3%) and 83/88 human PRs (94.3%).

3.2 Optimization Patch Characteristics (RQ1)

This analysis characterizes the optimization patterns applied in perf PRs authored by AI agents and humans. To enable systematic labeling, we adopt the catalog from SysLLMatic [34] (§2.1.1). This catalog organizes performance improvements at two levels of granularity: high-level categories for broad optimization strategies (e.g., Memory/Data Locality Optimization), and finer-grained patterns for specific implementation mechanisms (e.g., Caching).

3.2.1 General Labeling Procedure. We use LLMs as annotation aids and calibrate via iterative human inspection [41]. We use two commercial LLMs (SOTA as of December 2025): GPT-5.1 and GEMINI-3-PRO-PREVIEW, with temperature=0, to independently classify each patch. When the models agree, we accept the label, and when they disagree, two human annotators inspect the patch and adjudicate a final label [2]. Next, we describe how we refine the catalog and measure labeling error to contextualize the expected uncertainty.

3.2.2 Iterative Refinement. The SysLLMatic catalog was only applied to classify benchmark optimizations (e.g., HumanEval [8]). It had not been validated on real-world PRs, where code changes are larger and noisier. In an initial pass, LLM agreement was modest, with $\kappa = 0.53/0.48$ for agent PRs and $\kappa = 0.42/0.35$ for human PRs (category/pattern). Manual review revealed optimization strategies absent from the original catalog; thus, we refined it using PRs where the two models disagreed. Two authors (both Ph.D students) independently inspected these cases and revised the catalog iteratively until additional reviews produced no new patterns [11]. This process expanded the catalog from 43 to 59 patterns through 17 additions, 1 deletion, and 5 edits, and introduced two new high-level categories: *Build, Compilation, and Infrastructure Optimization* and *Network, Database, and Data Access Optimization* (see artifact).

With the revised catalog, LLMs agreement improved to $\kappa = 0.61/0.47$ for human PRs and $\kappa = 0.62/0.47$ for agent PRs (category/pattern). Remaining disagreement cases were manually labeled by two annotators, with an additional No Meaningful Performance

Change label for trivial or non-perf edits. Inter-annotator agreement was $\kappa = 0.66/0.58$ for human PRs and $\kappa = 0.60/0.60$ for agent PRs. Remaining disagreements were resolved through discussion to produce final consensus labels. We additionally measure error rate to assess reliability [39]. In LLM disagreement cases, alignment with final manual labels is limited (GPT: 35.9%/18.0%; Gemini: 38.0%/22.1% at category/pattern), improving to 48.3%/24.1% and 51.0%/29.7% after excluding 50 No Meaningful Performance Change cases. In LLM-agreed cases, manual auditing of a 10% random sample (20 PRs) indicates an error rate of 10%.

3.3 Testing and Evaluation Behavior (RQ2)

This question examines the practices used to measure performance (e.g., benchmarks, profiling) and how frequently such tests are reported. Because the AIDev dataset does not provide activity-level developer traces, our analysis captures how validation is described in pull request artifacts (*i.e.*, self-report).¹ We use claims of testing and evaluation as evidence that these activities were performed, but acknowledge that LLMs may not be trustworthy in this regard.

We analyze how performance validation is presented by examining PR descriptions, comments, and documentation, and classify such evidence using the three categories defined in §2.1.2: *Benchmark-Based Validation*, *Profiling-Based Validation*, and *Static-Reasoning-Based Validation*. To account for validation claims that lack quantitative and analytical support, we introduce a fourth category, *Anecdotal or Informal Local Testing*. PRs with no validation evidence are labeled as having no performance validation. Following RQ1, we exclude 50 PRs labeled as No Meaningful Performance Change, as validation claims are not meaningful for those PRs.

We use the same LLMs as in RQ1 to independently classify each PR, achieving agreement of $\kappa = 0.74$ for validation presence and $\kappa = 0.75$ for validation type. For the remaining PRs with model disagreement, two authors independently reviewed and adjudicated labels, with inter-annotator agreement of $\kappa = 0.62$ and $\kappa = 0.58$, respectively. We additionally measure error rates to assess reliability. In LLM disagreement cases, GPT aligns with the manual labels in 32.8% of validation presence and 12.1% of validation type, whereas Gemini aligns in 75.9% and 65.5%, respectively. In LLM-agreed cases, we manually audited 10% of cases (29 PRs). We observed error rates of 6.9% for validation presence and 10% for validation type.

4 Results

In this section, we first present a quantitative characterization of the dataset and then answer our two research questions. Some additional data visualizations are provided in the artifact (§6.2).

4.1 Initial Dataset Characterization

We begin by quantifying AI- and human-authored perf PRs along adoption outcomes (merge rate and time to merge), patch size, and maintainability. The result shows that agent perf PRs are merged less frequently than human ones (57% vs. 65%). However, when merged, agent PRs are integrated substantially faster, with a median time to merge of 0.03 hours, compared to 2.65 hours for human ones. Agent PRs cluster near immediate merge, while human PRs show a

¹Observing actual behavior would require fine-grained event data such as an IDE-instrumented dataset (e.g., the MSR 2018 Mining Challenge [35]).

broader, long-tailed distribution (Mann-Whitney U [29], $p < 0.001$). In terms of maintainability (Figure 2), AI- and human-authored PRs show substantial overlap across all three metrics (Total NLOC, AvgCCN [43], and Function Count), with no significant differences in their central distributions ($p > 0.05$). This indicates that the maintainability impact of agent PRs is comparable to that of human-authored patches. Similar fractions of AI and human PRs increase AvgCCN (40.14% vs. 41.94%), but agents show a heavier positive tail, indicating larger complexity increases when they occur.

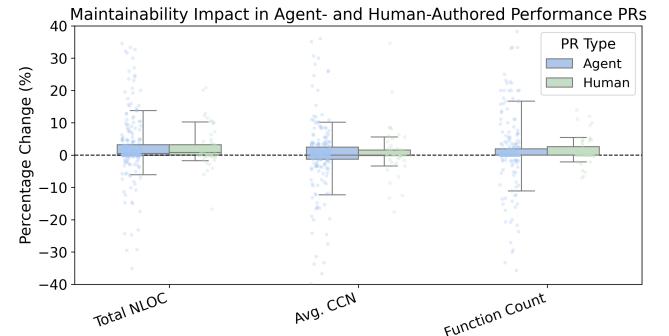


Figure 2: Patch size and maintainability measured with Lizard [38]. Boxes show medians and interquartile ranges, with whiskers extending to the 10th and 90th percentiles. AI-authored PRs show broadly similar medians to human ones, but exhibit higher dispersion and a larger number of extreme outliers.

4.2 Optimization Patch Characteristics (RQ1)

We analyze the optimization patterns applied by agents and humans across performance patches. Figure 3 compares the normalized distributions of high-level optimization patterns between agent and human PRs. It shows that the distributions are visually similar, with both groups emphasizing memory/data-locality and algorithm-level optimizations. Because both variables are categorical, we perform a chi-square test of independence [1, 24] on a 2×9 contingency table. The test finds no significant association between author type and optimization category ($\chi^2(8) = 6.10$, $p = 0.636$; Cramér's $V = 0.13$), indicating that agents and humans emphasize similar classes of high-level performance optimizations.

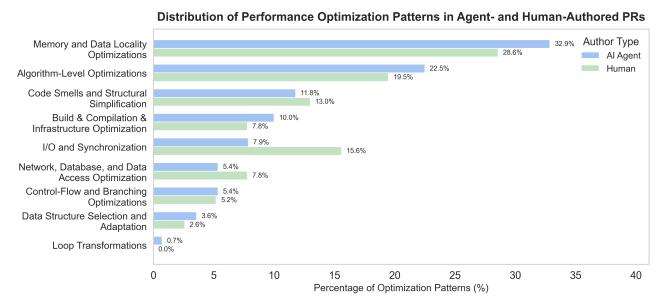


Figure 3: Distribution of high-level optimization patterns in AI- and human-authored perf PRs. Both groups exhibit similar distributions, with memory and data-locality optimizations dominating.

At a finer granularity, agent PRs exhibit a larger set of distinct optimization sub-patterns than human ones (38 vs. 21 of 59 total patterns); however, this difference reflects the larger volume of

agent PRs rather than systematically broader optimization behavior. To assess this, we measure sub-pattern richness and control for sample-size imbalance using a label-shuffling permutation test [14] and a subsampling-based rarefaction analysis [36]. Neither test finds a statistically significant difference between agent and human PRs (permutation $p = 0.12$; rarefaction $p = 0.13$), indicating that agents do not exhibit a wider variety of performance optimization strategies than humans; rather, they apply the same classes of optimizations observed in human-authored PRs.

4.3 Testing and Evaluation Behavior (RQ2)

We analyze whether perf PRs differ by author type in (i) the presence of validation, and (ii) the type of validation employed.

Validation Presence. Among agent PRs, 128 of 280 (45.7%) include validation, while 49 of 77 (63.6%) human-authored PRs include validation. A chi-square test of independence indicates a statistically significant but weak association between author type and validation presence ($\chi^2(1) = 7.06, p = 0.007$), with a Cramér's V of 0.14 [9].

Validation Type. We further analyze the distribution of validation methods, as shown in Figure 4. Agent-authored PRs predominantly rely on static-reasoning-based validation, which appears in 67.2% of validated AI PRs, compared to 44.9% for human PRs. In contrast, humans more frequently report benchmark-based validation, with 49.0% of validated human PRs including benchmark results, versus 25.0% for agents. A chi-square test indicates a statistically significant association between author type and validation method ($\chi^2(3) = 12.43, p = 0.006$), with a Cramér's V of 0.26.

These results indicate that agents report validation less frequently overall and rely more heavily on static reasoning, whereas humans more frequently use benchmark-based validation when reporting performance improvements. We then inspect individual PRs to illustrate the strengths and limitations of agentic validation practices. We observe that agents can add new benchmarks (e.g., inserting `time.perf_counter()` around the optimized call) and providing static reasoning based on algorithmic complexity (e.g., replacing string concatenation with `StringBuilder` to reduce complexity from $O(n^2)$ to amortized $O(n)$). However, they may also report benchmark data without supporting evidence (e.g., 7400× speedup stated in the PR description without corresponding benchmark code), exposing such claims to the risk of hallucination.

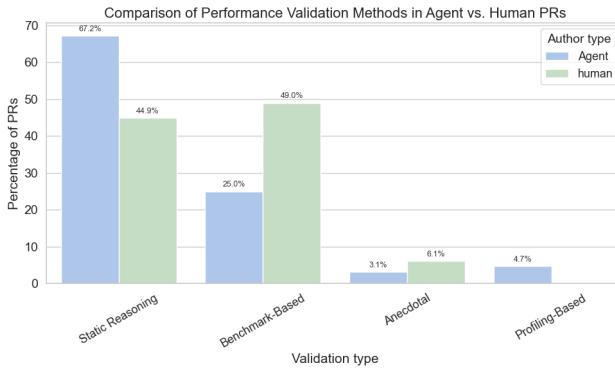


Figure 4: Distribution of types of validation reported in perf PRs. Agent PRs rely predominantly on static reasoning checks, whereas human PRs more frequently report benchmark-based validation in addition to static reasoning.

5 Threats to Validity

We discuss construct, internal, and external threats to validity [45].

Construct: We inspect PR artifacts for validation evidence; validation conducted outside the PR workflow or documented informally may lead us to underestimate performance validation practices. We use LLM-assisted annotation to classify optimization patterns and detect validation evidence in PRs, which may be impacted by hallucinations or misinterpretation of code. We mitigate this by using temperature = 0, two independent LLMs, and Cohen's κ for inter-model agreement; disagreements are resolved by two human annotators, though some subjectivity may remain.

Internal: We use chi-square, permutation, and rarefaction tests. Chi-square assumes independent observations and adequate cell counts, which may be violated by repository/author clustering and sparse categories. Permutation and rarefaction mitigate sample-size imbalance but assume label exchangeability. Significant results with small effect sizes should be interpreted as associative, not causal.

External: Our study builds on the AIDev dataset and inherits its limitations, including potential PR author-labeling errors and its asymmetric repository filters for agent vs. human PRs (> 100 vs. > 500 stars), which can bias the sample composition and size.

6 Discussion

6.1 Towards Next-Gen Code Optimization

Our analysis shows that agent PRs draw from a similar optimization pattern space as human ones (§4.2). This alignment highlights both the effectiveness of current LLMs in acquiring established performance engineering knowledge and the limits of their present behavior in practice. Despite decades of work showing the effectiveness of loop techniques such as unrolling [37] and fusion [23], they remain the least frequently applied category: they are absent from human perf PRs and appear in only 0.7% of agent-authored ones (Figure 3). In human development workflows, loop techniques are often avoided because they can reduce code readability and maintainability, and are error-prone to implement correctly [25, 26].

These constraints do not apply in the same way to AI agents. As software development becomes increasingly agent-centric, this gap highlights an underexplored space: enabling agents to consider optimization techniques that are traditionally underutilized by human. Realizing this potential requires agents tightly integrated with profiling feedback, correctness checks, and automated evaluation pipelines, allowing them to explore a broader optimization space while maintaining the reliability expected of production systems.

6.2 Performance Validation Gaps in Agent PRs

Estimating performance impact is challenging – sensitivity to workloads, hardware, and measurement noise often produces misleading or irreproducible results [28, 32]. Thus, rigorous improvements require validation through benchmarking or profiling [6, 17]. Despite this, we found that agent perf PRs include explicit testing or validation substantially less frequently than human ones (§4.3). In the absence of explicit requirements or safeguards, this gap increases the risk that agent-generated performance changes are integrated without sufficient empirical evidence. We further observe a difference in validation style: agents rely on static-reasoning-based

justification, whereas humans more often provide benchmark-based evidence (Figure 4). Although static reasoning can indicate plausible performance effects, it lacks quantitative grounding and is therefore susceptible to incorrect assumptions or hallucinated claims.

As agent-authored perf PRs grow, future work should consider how agents access and invoke performance evaluation infrastructure, since existing CI-based benchmarking and testing resources, designed around human contribution rates, are unlikely to scale to agent-driven workloads. Addressing this mismatch requires standardized benchmarking services and shared profiling resources, and community-level platforms for automated performance validation.

Data Availability. Our script, data, and prompt are available at: https://anonymous.4open.science/r/perf_patch_study.

References

- [1] Alan Agresti. 2013. *Categorical Data Analysis* (3rd ed.). Wiley, Hoboken, NJ, USA.
- [2] Toufique Ahmed, Premkumar Devanbu, Christoph Treude, and Michael Pradel. 2025. Can llms replace manual annotation of software engineering artifacts?. In *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*. IEEE, 526–538.
- [3] David A. Bader, Bernard M. E. Moret, and Peter Sanders. 2002. *Algorithm Engineering for Parallel Computation*. Springer Berlin Heidelberg.
- [4] Prasanna Balaprakash, Jack Dongarra, Todd Gamblin, Mary Hall, Jeffrey K Hollingsworth, Boyana Norris, and Richard Vuduc. 2018. Autotuning in high-performance computing applications. *Proc. IEEE* 106, 11 (2018), 2068–2083.
- [5] Abhijeet Banerjee. 2014. Static analysis driven performance and energy testing. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (Hong Kong, China) (FSE 2014)*. Association for Computing Machinery, New York, NY, USA, 791–794. doi:10.1145/2635868.2666602
- [6] Dirk Beyer, Stefan Löwe, and Philipp Wendler. 2019. Reliable benchmarking: requirements and solutions. *Int. J. Softw. Tools Technol. Transf.* 21, 1 (Feb. 2019), 1–29. doi:10.1007/s10009-017-0469-y
- [7] Rainer Buechi, Vincent Heuvelink, Karl, et al. 2012. A survey on hardware-aware and heterogeneous computing on multicore processors and accelerators. *Concurrency and Computation: Practice and Experience* (2012). doi:doi:abs/10.1002/cpe.1904
- [8] Mark Chen, Jerry Tworek, Heewoo Jun, et al. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG] <https://arxiv.org/abs/2107.03374>
- [9] Jacob Cohen. 2013. *Statistical power analysis for the behavioral sciences*. routledge.
- [10] Spandan Garg, Roshanak Zilouchian Moghadam, and Neel Sundaresan. 2025. RAPGen: An Approach for Fixing Code Inefficiencies in Zero-Shot. arXiv:2306.17077 [cs.SE] <https://arxiv.org/abs/2306.17077>
- [11] Barney G. Glaser and Anselm L. Strauss. 1967. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine Publishing Company, Chicago, IL.
- [12] Stefan Goedecker and Adolfy Hoisie. 2001. *Performance optimization of numerically intensive codes*. SIAM.
- [13] Jingzhi Gong, Vardan Voskanyan, Paul Brookes, et al. 2025. Language Models for Code Optimization: Survey, Challenges and Future Directions. arXiv:2501.01277 [cs.SE] <https://arxiv.org/abs/2501.01277>
- [14] Phillip Good. 2013. *Permutation tests: a practical guide to resampling methods for testing hypotheses*. Springer Science & Business Media.
- [15] Brendan Gregg. 2016. The flame graph. *Commun. ACM* 59, 6 (2016), 48–57.
- [16] Brendan Gregg. 2021. *Systems Performance: Enterprise and the Cloud*. Addison-Wesley.
- [17] Torsten Hoefer and Roberto Belli. 2015. Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (Austin, Texas) (SC ’15)*. Association for Computing Machinery, New York, NY, USA, Article 73, 12 pages. doi:10.1145/2807591.2807644
- [18] Vojtěch Horký, Peter Libič, Lukáš Marek, Antonín Steinhauser, and Petr Tůma. 2015. Utilizing Performance Unit Tests To Increase Performance Awareness. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (Austin, Texas, USA) (ICPE ’15)*. Association for Computing Machinery, New York, NY, USA, 289–300. doi:10.1145/2668930.2688051
- [19] Raj Jain. 1991. *The Art of Computer Systems Performance Analysis: Techniques For Experimental Design, Measurement, Simulation, and Modeling*. Wiley-Interscience.
- [20] Mohammad Talal Jamil, Shamsa Abid, and Shafay Shamail. 2025. Can LLMs Generate Higher Quality Code Than Humans? An Empirical Study. In *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*. 478–489. doi:10.1109/MSR66628.2025.00081
- [21] Herb Krasner. 2021. The cost of poor software quality in the US: A 2020 report. *Proc. Consortium Inf. Softw. QualityTM (CISQTM)* 2 (2021), 3.
- [22] Hao Li, Haoxiang Zhang, and Ahmed E. Hassan. 2025. The Rise of AI Teammates in Software Engineering (SE) 3.0: How Autonomous Coding Agents Are Reshaping Software Engineering. *arXiv preprint arXiv:2507.15003* (2025).
- [23] N. Manjikian and T.S. Abdelrahman. 1997. Fusion of loops for parallelism and locality. *IEEE Transactions on Parallel and Distributed Systems* 8, 2 (1997), 193–209. doi:10.1109/71.577265
- [24] Mary L McHugh. 2013. The chi-square test of independence. *Biochemia medica* 23, 2 (2013), 143–149.
- [25] Kathryn S. McKinley, Steve Carr, and Chau-Wen Tseng. 1996. Improving data locality with loop transformations. *ACM Trans. Program. Lang. Syst.* 18, 4 (July 1996), 424–453. doi:10.1145/233561.233564
- [26] T. Mens and T. Tourwe. 2004. A survey of software refactoring. *IEEE Transactions on Software Engineering* 30, 2 (2004), 126–139. doi:10.1109/TSE.2004.1265817
- [27] Steven S. Muchnick. 1998. *Advanced compiler design and implementation*. Morgan Kaufmann Publishers Inc.
- [28] Todd Mytkowicz, Amer Diwan, Matthias Hauswirth, and Peter F. Sweeney. 2009. Producing wrong data without doing anything obviously wrong!. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (Washington, DC, USA) (ASPLOS XIV)*. Association for Computing Machinery, New York, NY, USA, 265–276. doi:10.1145/1508244.1508275
- [29] Nadim Nachar et al. 2008. The Mann-Whitney U: A test for assessing whether two independent samples come from the same distribution. *Tutorials in quantitative Methods for Psychology* 4, 1 (2008), 13–20.
- [30] Oswaldo Olivo, Isil Dillig, and Calvin Lin. 2015. Static detection of asymptotic performance bugs in collection traversals. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation (Portland, OR, USA) (PLDI ’15)*. Association for Computing Machinery, New York, NY, USA, 369–378. doi:10.1145/2737924.2737966
- [31] I. Ozkaya. 2023. Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications. *IEEE Software* (2023).
- [32] Alessandro Vittorio Papadopoulos, Laurens Versluis, André Bauer, Nikolas Herbst, Joakim von Kistowski, Ahmed Ali-Eldin, Cristina L. Abad, José Nelson Amaral, Petri Túma, and Alexandru Iosup. 2021. Methodological Principles for Reproducible Performance Evaluation in Cloud Computing. *IEEE Transactions on Software Engineering* 47, 8 (2021), 1528–1543. doi:10.1109/TSE.2019.2927908
- [33] Huiyun Peng, Arjun Gupta, Nicholas John Eliopoulos, Chien Chou Ho, Rishi Mantri, Leo Deng, Wenxin Jiang, Yung-Hsiang Lu, Konstantin Läufer, George K. Thiruvathukal, and James C. Davis. 2024. Large Language Models for Energy-Efficient Code: Emerging Results and Future Directions. arXiv:2410.09241 [cs.SE] <https://arxiv.org/abs/2410.09241>
- [34] Huiyun Peng, Arjun Gupta, Ryan Hasler, Nicholas John Eliopoulos, Chien-Chou Ho, Rishi Mantri, Leo Deng, Konstantin Läufer, George K. Thiruvathukal, and James C. Davis. 2025. SysLLMatic: Large Language Models are Software System Optimizers. arXiv:2506.01249 [cs.SE] <https://arxiv.org/abs/2506.01249>
- [35] Sebastian Proksch, Sven Amann, and Sarah Nadi. 2018. Enriched Event Streams: A General Dataset for Empirical Studies on In-IDE Activities of Software Developers. In *Proceedings of the 15th Working Conference on Mining Software Repositories*.
- [36] Howard L. Sanders. 1968. Marine benthic diversity: a comparative study. *The American Naturalist* 102, 925 (1968), 243–282.
- [37] Vivek Sarkar. 2000. Optimized unrolling of nested loops. In *Proceedings of the 14th International Conference on Supercomputing (Santa Fe, New Mexico, USA) (ICS ’00)*. Association for Computing Machinery, New York, NY, USA, 153–166. doi:10.1145/335231.335246
- [38] terryyin. 2025. lizard. <https://github.com/terryyin/lizard>. Accessed: 2025-12-19.
- [39] Petter Törnberg. 2024. Best practices for text annotation with large language models. *arXiv preprint arXiv:2402.05129* (2024).
- [40] Rosalia Tufano, Antonio Mastropaoletti, Federica Pepe, Ozren Dabic, Massimiliano Di Penta, and Gabriele Bavota. 2024. Unveiling ChatGPT’s Usage in Open Source Projects: A Mining-based Study. In *Proceedings of the 21st International Conference on Mining Software Repositories (Lisbon, Portugal) (MSR ’24)*. Association for Computing Machinery, New York, NY, USA, 571–583. doi:10.1145/3643991.3644918
- [41] Xinru Wang, Hannah Kim, Sajjadur Rahman, Kushan Mitra, and Zhengjie Miao. 2024. Human-Ilm collaborative annotation through effective verification of Ilm labels. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–21.
- [42] Miku Watanabe, Hao Li, Yutaro Kashiwa, Brittany Reid, Hajimu Iida, and Ahmed E Hassan. 2025. On the use of agentic coding: An empirical study of pull requests on github. *arXiv preprint arXiv:2509.14745* (2025).
- [43] Arthur Henry Watson, Dolores R Wallace, and Thomas J McCabe. 1996. *Structured testing: A testing methodology using the cyclomatic complexity metric*. Vol. 500. US Department of Commerce, Technology Administration, National Institute of
- [44] Claas Wilke, Sebastian Richly, Sebastian Götz, Christian Piechnick, and Uwe Alßmann. 2013. Energy Consumption and Efficiency in Mobile Applications: A

- User Feedback Study. In *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*. 134–141. doi:10.1109/GreenCom-iThings-CPSCo.2013.45
- [45] Claes Wohlin, Per Runeson, Hst, et al. 2012. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated.
- [46] Lirong Yi, Gregory Gay, and Philipp Leitner. 2025. An Experimental Study of Real-Life LLM-Proposed Performance Improvements. arXiv:2510.15494 [cs.SE] <https://arxiv.org/abs/2510.15494>