Do LLMs Suggest Consistent Identifiers? An Empirical Study on GitHub Pull Requests

Julyanara R. Silva
Instituto Federal do Triângulo Mineiro (IFTM)
Uberlândia, Brazil
julyanara.silva@estudante.iftm.edu.br

Marcelo A. Maia
Universidade Federal de Uberlândia (UFU)
Uberlândia, Brazil
marcelo.maia@ufu.br

Carlos Eduardo C. Dantas
Instituto Federal do Triângulo Mineiro (IFTM)
Uberlândia, Brazil
carloseduardodantas@iftm.edu.br

ABSTRACT

The appropriate naming of identifiers is crucial in the source code, as names can represent up to 70% of all characters and play a key role in helping developers understand the intent behind the code. Renaming is also among the most common refactoring operations performed by developers. Although static analysis tools can detect violations of naming conventions, there remains a gap in tools capable of suggesting names that are semantically aligned with the purpose of the code. In this study, we collected 152 Java and 168 Python instances of GitHub pull requests (PRs) that improved the identifier names. We investigated whether Large Language Models (LLMs) are capable of suggesting names that are semantically consistent with those chosen by developers. We also compared three temperature settings, ranging from deterministic to creative, to assess how this parameter influences naming suggestions. Our results show that ChatGPT slightly outperforms the other models for Python identifiers, exactly matching the developers' chosen names in 16.2% of the cases and achieving high semantic similarity in 65.1%. Overall, the three LLMs performed better in Python than in Java. Regarding the temperature setting, the moderate temperature yielded slightly better results in Python, while the creative setting performed slightly better in Java. These findings suggest that tasks such as naming identifiers may benefit from different parameter configurations than those typically used in more deterministic tasks such as code refinement.

KEYWORDS

Code identifiers, LLM, ChatGPT, DeepSeek, Gemini, Pull Request

1 Introduction

Identifiers are a fundamental part of software, because they communicate the intent behind their use and represent approximately 70% of all characters in the source code [6]. Improving identifiers is one of the most common activities in refactoring operations performed by developers in real-world repositories [14], as it helps make the code easier to read [3] and understand [13]. According to Robert C. Martin, identifiers are responsible for 90% of the readability of the software [11]. Given their importance, several studies have proposed approaches to recommend more appropriate identifiers for code elements [1, 10, 21].

Although developers frequently modify identifiers, as observed in Github repositories [3, 14], they are not often supported by automated tools to suggest appropriate names. For example, static analysis tools such as SonarLint [20] typically notify users when a name violates language-specific naming conventions (e.g., using the *camelCase* pattern or starting variable names with lowercase letters in Java). However, these tools generally overlook semantic aspects, such as whether the chosen name accurately reflects the behavior of the code.

As Large Language Models (LLMs) have been gaining increasing attention from the software development community, several studies have proposed evaluating their effectiveness in generating code for various tasks, including refactoring operations [23], bug fixing [19], suggestions of code changes, and other programming activities on GitHub [18, 22]. ChatGPT, in particular, has also shown potential in supporting code comprehension [12] and producing code snippets with identifiers using appropriate Java conventions [4].

In this work, we collected 320 instances from 193 GitHub merged Pull Requests (PRs) focused on improving identifiers to evaluate to what extent LLMs are capable of improving the names according to human perception. This study provides the following contributions.

(1) A comparative study between the identifiers given by LLMs such as ChatGPT 4.1, Gemini 2.5 Pro and DeepSeek coder, a model trained on coding generation.
(2) An analysis of the impact of the temperature parameter on the identifier suggestion in Java and Python code.
(3) A replication package that includes scripts to use LLM, queries in Github, and the corresponding data to support future research [9].

2 Methodology

This study is driven by the following research questions:

RQ$_1$: Can LLMs suggest identifiers that align with human perception? This research question investigates whether the identifiers suggested by LLMs are consistent with those improved by developers in real-world GitHub repositories through merged PR, i.e., a name that was approved in a code review process.

RQ$_2$: Does the temperature parameter influence the quality of LLM identifier suggestions? Previous work has shown that lower temperature values often lead to better and more stable results in code refinement tasks [8]. However, this assumption may not be generalized to all scenarios involving source code. For example, suggesting variable names with LLMs may benefit from exploring a wider range of temperature settings, as naming often requires a degree of creativity to align semantically with the intent of the code.

RQ$_3$: Does the effectiveness of the LLM suggested identifiers vary between Python and Java? This research question investigates whether LLMs are more successful in suggesting identifiers that align with those improved by developers and accepted

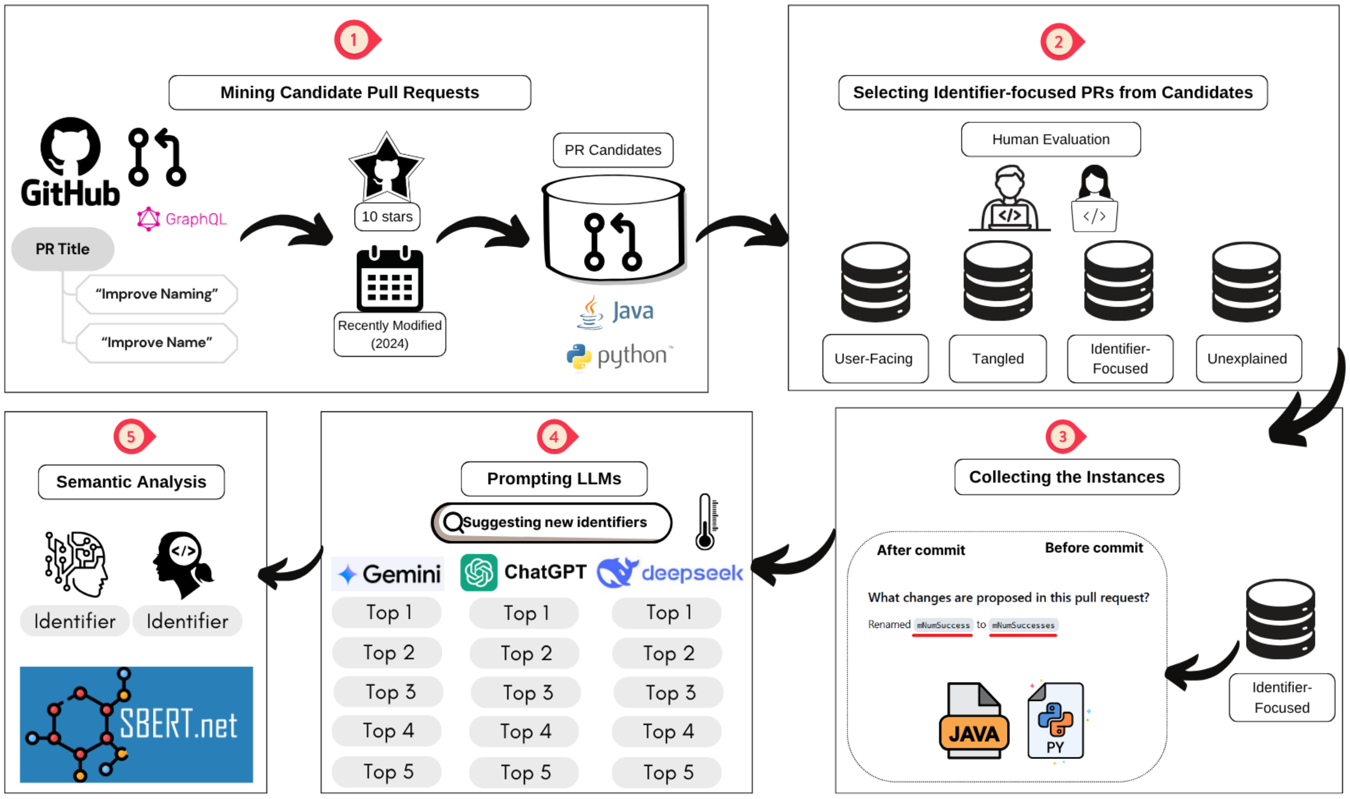


Figure 1: Overview of the proposed methodology

by GitHub reviewers in real-world GitHub repositories, depending on the programming language.

Figure 1 illustrates the methodology proposed to answer the research questions. Each step is described in the following subsections.

2.1 Mining Candidate Pull Requests

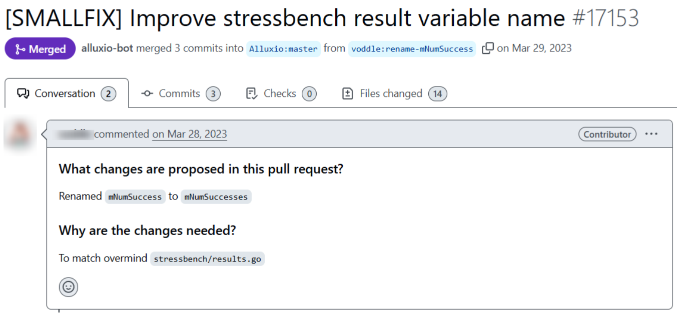
To identify merged PRs focused on improving identifiers, a query was executed in May 2025 using the GitHub GraphQL API [7]. The analysis focused on PRs that included one of the following keywords: "improve naming", or "improve name" in their title field, since the title could indicate the motivation behind the PR. This query also excluded GitHub repositories with fewer than 10 stars. This filtering criterion has also been adopted in previous studies [13, 22]. PRs were also excluded if they originated from forked repositories or repositories that did not have an active code review process, defined as having no merged PRs in 2024. Furthermore, PRs were also excluded if they did not modify Python or Java files (e.g., those improving textual files such as *README*, *.md* and *.rst* files) and excluded PRs that did not have at least two different participants (the PR author and a reviewer) to ensure a peer review process. Finally, PRs were removed if the text is written in the non-English language. After applying these criteria, 511 candidate PR from Python and 358 candidates from Java repositories were identified, in a total of 869 PR candidates. This process is illustrated in step 1 of Figure 1.

2.2 Selecting Identifier-Focused PRs from Candidates

In this subsection, the objective is to select merged PR that were created exclusively to improve the identifiers. The main motivation for this filtering is based on the assumption that a PR focused on renaming, approved through a peer review process, could contain identifiers that are semantically aligned with the actual behavior of the code.

Although the 511 candidate PR in Python and 358 in Java mention identifier-related keywords in their titles, some false positives were identified, that is, PR that contain such keywords but not actually focused on improving identifiers. Therefore, the evaluators manually inspected the PR title and body fields of each candidate PR into four categories as shown in step 2 of Figure 1.

(1) User-Facing PR - Although identifier keywords are mentioned, the PR is focused on improving other naming aspects, such as console output, log messages, colors, styles, and others, rather than focusing on improving the identifier for other developers to have easier comprehension.
(2) Tangled PR - In this case, the PR improves code identifiers but also includes other changes, such as adding new features, fixing bugs, or other types of refactoring operations.
(3) Unexplained PR - Some PR mention identifier improvements in vague terms, without clearly explaining what was actually performed.



[SMALLFIX] Improve stressbench result variable name #17153

Merged alluxio-bot merged 3 commits into `Alluxio:master` from `voddle:rename-mNumSuccess` on Mar 29, 2023

[redacted] commented on Mar 28, 2023 Contributor

What changes are proposed in this pull request?

Renamed `mNumSuccess` to `mNumSuccesses`

Why are the changes needed?

To match overmind `stressbench/results.go`

Figure 2: Example Identifier-Focused PR

(4) Identifier-focused PR — These PRs are explicitly aimed at improving identifiers, developers clearly describe such intentions in the PR title or body.

This process was performed by two evaluators, aiming to extract the *Identifier-focused* PRs, as an example shows in Figure 2. The *PR title* is "Improve stressbench result variable name," and in the *PR body*, the developer proposes renaming `mNumSucess` to `mNumSucesses`. As a result of this filtering process, we selected 93 Python PRs and 98 Java *Identifier-focused* PRs, totaling 191 PRs.

2.3 Collecting the Instances

In this step, for each of the 93 Python and 98 Java *Identifier-focused* PRs identified in the previous step, we extracted the identifiers mentioned by the developers, along with the classes where they were defined. For example, in Figure 2, although the PR modified 14 files, the identifier `mNumSuccess` is a field of the class `TaskResultStatistics`.

At this stage, for each PR, we collected: (i) the name of the identifier before the commit, (ii) the name of the identifier after the commit, and (iii) the file in which the identifier was originally defined and later renamed. In total, we extracted 320 instances, 152 from Java PRs and 168 from Python PRs, as a single PR may rename multiple identifiers. These identifiers are classified as 207 variables, 67 methods, 19 classes, 16 fields, and 11 constants.

2.4 Prompting LLMs

In this process, each of the 320 improved identifiers found in merged GitHub PRs was used as input for the LLMs, which were tasked with generating five alternative identifier names. The choice of providing five suggestions was inspired by previous work showing that developers typically inspect around 3.41 code search results per session on average [17]. A script was created to construct the prompts and extract these names using the *openai API* for ChatGPT 4.1 and the DeepSeek coder (a model specialized in coding-related tasks [5]) versions, and the *google.generativeai API* for the Gemini 2.5 Pro version. All using the parameter *MAX_TOKENS* = 8,192, that is, each request is limited to this number of tokens. and three *TEMPERATURE* parameters, that is, 0.0 (deterministic), 1.0 (moderate) and 2.0 (creative).

The prompt used to ask the LLM to suggest identifier names was built using a context defined in the system prompt, as shown in the text box below. In this prompt, `[PROGRAMMING_LANGUAGE]` refers to Java or Python, `[IDENTIFIER]` represents the original name of the identifier before it was changed by the developer, and `[CODE_SNIPPET]` contains the code where the identifier was declared, providing the LLM with context about the purpose of that identifier.

2.5 Analysing Semantic Similarity

The top-5 identifier names suggested by each LLM are extracted, and their semantic similarity is computed by converting the identifiers into word embeddings and comparing them with those implemented by developers in merged GitHub PRs. To perform this comparison, *Sentence Transformers* were used, [15] specifically the *all-MiniLM-L6-v2* model, which has also been adopted in previous studies [2]. Sentence Transformers is a Python framework that converts text, images, and sentences into embeddings for multiple languages, enabling the measurement of similar meanings using cosine similarity. This technique is useful in a variety of tasks, such as mining a paraphrase and semantic search [16].

The *Sentence Transformers* produce a similarity score ranging from -1.0 to +1.0, where -1.0 indicates completely opposite meanings and +1.0 indicates identical meaning. Thus, the higher the score, the greater the semantic similarity between the identifiers.

Prompt Template for label readability improvements

System: "You are a software developer, with expertise in suggesting identifiers that accurately reflect the behavior and intent of the code."
User: "Given the following `[PROGRAMMING_LANGUAGE]` code: `[CODE_SNIPPET]`, please review the identifier `[IDENTIFIER]` and suggest five alternative names that better align with its purpose in the code. List your naming as follows, without explanation, only the identifiers names:
Top 1 – [most suitable name]
Top 2 – [second best name]
Top 3 – [third best name]
Top 4 – [fourth best name]
Top 5 – [fifth best name]"

3 Results

Table 1 presents the results grouped by temperature, model, and programming language. From the five identifier suggestions generated by each model, we selected the one with the highest score computed by *Sentence Transformers*. We report the number of exact or near matches using the HIT metric, which counts how often a model's top suggestion aligns with the developer's choice. For example, under the Hit@1.0 threshold, ChatGPT with temperature 1.0 (moderate) exactly matched the developer's identifier in 27 out of 168 Python instances. In Java, Gemini with creative temperature (2.0) achieved 22 exact matches out of 152 instances.

Using the broader threshold of HIT > 0.7, ChatGPT with the creative temperature produced semantically similar identifiers in Python 108 instances. Although scores between 0.7 and 0.99 do not represent an exact match, they indicate strong semantic alignment. For example, in PR ID = 7864 from the *apache/iceberg* repository, the developer renamed the method from `testCreateCreateCatalogHive()` to `testCreateCatalogHive()`. ChatGPT's top-1 suggestion was `testCreateHiveCatalog()`, which is semantically close

Table 1: Hit rate by temperature, model and programming language

Temperature	Language	Model	Hit 1.0	Hit > 0.9	Hit > 0.8	Hit > 0.7
0.0	Python	Chatgpt	26 (15.5%)	54 (32.1%)	**84 (50.0%)**	104 (61.9%)
0.0	Python	Deepseek	16 (9.5%)	34 (20.2%)	67 (39.9%)	98 (58.3%)
0.0	Python	Gemini	26 (15.5%)	43 (25.6%)	71 (42.3%)	100 (59.5%)
1.0	Python	Chatgpt	**27 (16.2%)**	**58 (34.7%)**	**84 (50%)**	107 (64.1%)
1.0	Python	Deepseek	17 (10.2%)	38 (22.8%)	70 (41.9%)	96 (57.5%)
1.0	Python	Gemini	24 (14.4%)	43 (25.7%)	70 (41.9%)	104 (62.3%)
2.0	Python	Chatgpt	19 (11.4%)	46 (27.7%)	77 (46.4%)	**108 (65.1%)**
2.0	Python	Deepseek	11 (6.6%)	34 (20.5%)	63 (38.0%)	93 (56.0%)
2.0	Python	Gemini	19 (11.4%)	38 (22.9%)	65 (39.2%)	93 (56.0%)
0.0	Java	Chatgpt	17 (11.2%)	32 (21.1%)	51 (33.6%)	77 (50.7%)
0.0	Java	Deepseek	11 (7.2%)	30 (19.7%)	47 (30.9%)	74 (48.7%)
0.0	Java	Gemini	18 (11.8%)	38 (25.0%)	51 (33.6%)	70 (46.1%)
1.0	Java	Chatgpt	16 (10.5%)	31 (20.4%)	53 (34.9%)	76 (50.0%)
1.0	Java	Deepseek	12 (7.9%)	31 (20.4%)	53 (34.9%)	77 (50.7%)
1.0	Java	Gemini	**22 (14.5%)**	**40 (26.3%)**	51 (33.6%)	71 (46.7%)
2.0	Java	Chatgpt	15 (10.0%)	33 (22.0%)	**55 (36.7%)**	**78 (52.0%)**
2.0	Java	Deepseek	12 (8.0%)	32 (21.3%)	49 (32.7%)	72 (48.0%)
2.0	Java	Gemini	**22 (14.5%)**	36 (24.0%)	53 (35.3%)	69 (46.0%)

to the developer's choice. The similarity score computed by *Sentence Transformers* for this suggestion was 0.93.

3.1 LLMs

When analyzing the performance of the LLMs, ChatGPT consistently achieved more hits in Python. In Java, both Gemini and ChatGPT outperformed DeepSeek. Figures 3, 4 and 5 present boxplots that illustrate the distribution of similarity scores produced by the LLMs. Although no statistically significant differences were found, ChatGPT slightly outperforms the other models, especially in Python, where it achieves the highest median similarity (close to 0.9 in all temperatures) and a compact distribution with few low outliers. In contrast, DeepSeek shows more variability, with lower median values and a higher number of outliers, particularly in Java, where it delivers the weakest performance among all models. Gemini demonstrates moderate performance, slightly better than DeepSeek in Java, but still less consistent than ChatGPT.

3.2 Programming Languages

To compare Python scores with Java scores, we selected the best score for each programming language per instance. Using the Mann-Whitney U test, we found statistically significant differences (p-value < 0.05) between Python and Java in deterministic and moderate temperature settings, but no significant differences in the creative setting. This suggests that, with a creative temperature, LLMs generated more semantically aligned names in Java. Overall, LLMs performed better in Python than in Java, indicating that Python contexts may be more conducive to generating name suggestions that better reflect the intent of the developer.

3.3 Temperature Parameter

When analyzing the temperature settings, although the variation among them is not substantial, the 1.0 (moderate) setting generally produced slightly better results for Python. For Java, the 2.0 (creative) settings perform better, as shown in Figure 5. This is a significant finding as it highlights that the optimal temperature value depends on the specific task. Although prior studies have shown that lower, deterministic temperatures tend to perform better in tasks such as code refinement [8], this does not appear to be the case in the context of identifier naming.

4 Threats to Validity

One construct threat to validity is the lack of guarantees that the identifier chosen by the developer is accurate in representing the meaning of the code. To mitigate this threat, we selected only merged PRs explicitly focused on improving naming, in which the changes were validated by both the author and the reviewer.

Another threat is the generalizability of our findings, a possibility due to false negatives. Although our methodology includes rigorous procedures to eliminate false positives (i.e., samples that are not focused on naming improvements), we cannot guarantee that all relevant cases were captured. Developers may open Github PRs to improve names using a variety of different keywords, making it difficult to identify and include all such cases in our dataset.

5 Related Work

Alsaedi et al. [2] used Sentence Transformers to compute the semantic similarity between bug report descriptions and method names, in order to identify which methods were more likely to be related to the reported bug. Their approach involved a two-step process: first, identifying the most likely buggy classes, then ranking the methods within those classes based on semantic similarity. The results showed that the use of semantic embeddings helped improve the accuracy of method-level bug localization. This work also uses Sentence Transformers, but with a different objective, that is, to measure the semantic similarity between identifier names suggested by the LLMs and those improved by developers in merged GitHub merged PRs.

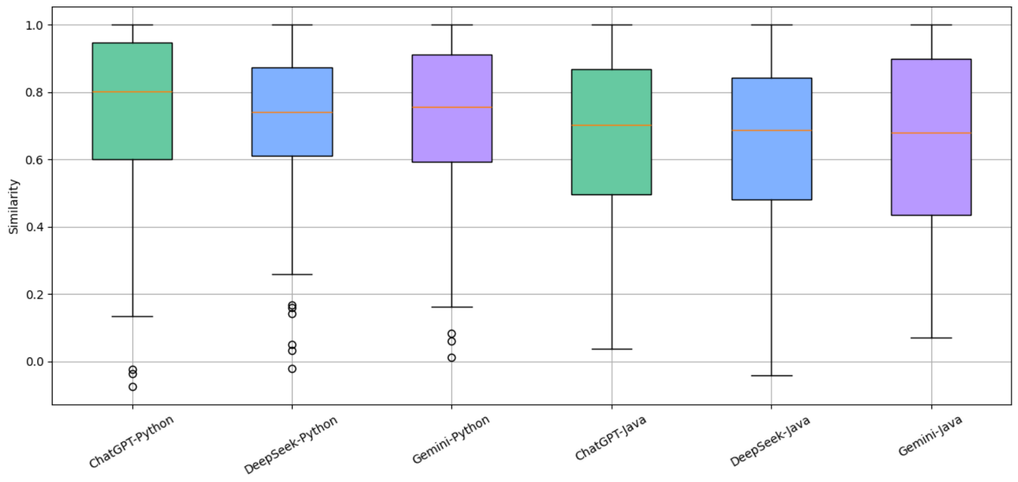


Figure 3: Distribution of the similarity scores using deterministic (0.0) temperature parameter

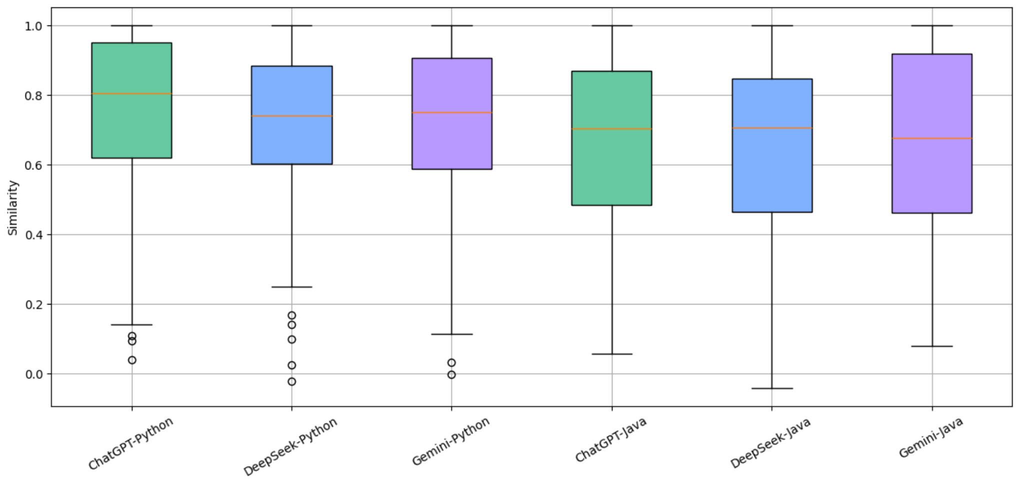


Figure 4: Distribution of the similarity scores using moderate (1.0) temperature parameter

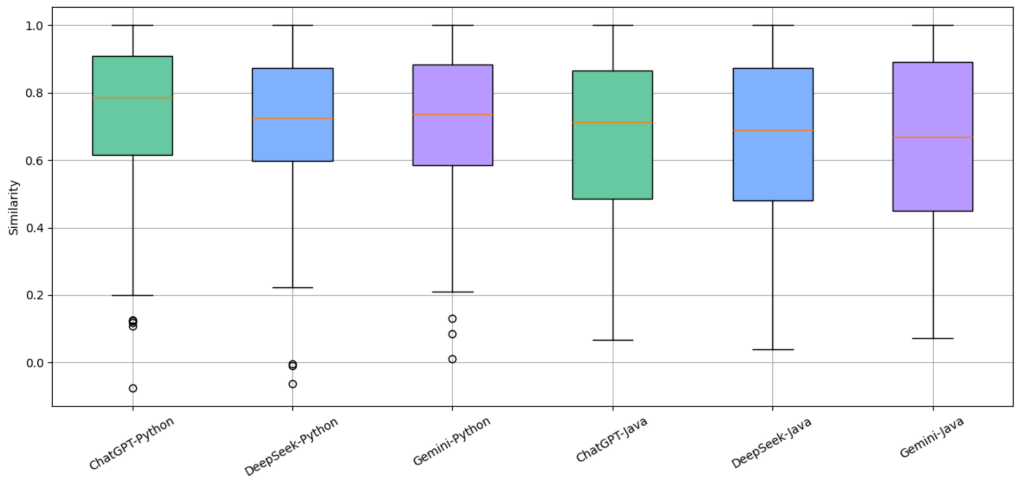


Figure 5: Distribution of the similarity scores using creative (2.0) temperature parameter

Lin et al. [10] proposed DeepName, a deep learning-based approach for recommending identifier names in source code. Their model captures the lexical and syntactic context surrounding an identifier and predicts a semantically appropriate name. Trained on a dataset of Java projects from GitHub, DeepName outperformed traditional baselines such as n-gram and frequency-based models. While their approach also focuses on identifier name recommendation, our work differs in that we rely on names actually adopted by developers in real-world GitHub PRs as the basis for evaluation.

Several previous studies have shown that developers are concerned with improving identifier names. For example, Pantiuchina et al.[14] proposed a taxonomy of refactoring operations, some of which are directly related to naming improvements. Similarly, Dantas et al.[3] developed a catalog of change types that leads developers to improve code readability, with naming improvements being the most common type of change observed. Although our study does not aim to produce a catalog of developer-performed operations, previous work highlights the central role of naming in code quality and readability.

6 Conclusion

This study investigated the semantic similarity between the identifier recommended by LLMs and those improved by developers in real-world GitHub PRs. By analyzing 320 instances (168 in Python and 152 in Java) and evaluating the top-5 identifier suggestions generated by each model, we obtained the key findings.

(1) ChatGPT slightly outperformed the other models in Python, achieving the highest exact match rates and the highest number of semantically aligned suggestions (e.g., 108 suggestions with similarity > 0.7 using the creative setting). In Java, both ChatGPT and Gemini performed better than DeepSeek, and Gemini achieved the highest number of exact matches (22) when using the creative temperature.
(2) RQ_1 answer: All LLMs have Hit > 0.7 in more than 50% of the Python samples, indicating that, in many instances, LLMs can suggest identifiers aligned with human perception.
(3) RQ_2 answer: The temperature parameter has some effect on the performance. For Python, the moderate setting (1.0) yielded slightly better overall results. In contrast, the creative setting (2.0) showed better performance in Java, particularly for ChatGPT and Gemini.
(4) RQ_3 answer: All LLMs performed better in Python than in Java, suggesting that Python code contexts may be easier for LLMs to interpret in terms of naming intent.

In future work, we plan to use a larger sample, including more keywords, programming languages, and separate the analysis by types of identifiers, such as field, variable, function, class name, and others.

ARTIFACT AVAILABILITY

The replication package is available at [9]

ACKNOWLEDGMENTS

This work was supported by the BIC JR program of the Instituto Federal do Triângulo Mineiro (IFTM), which provided a research scholarship to one of the authors.

REFERENCES

[1] Miltiadis Allamanis, Earl T. Barr, Christian Bird, and Charles Sutton. 2014. Learning natural coding conventions. 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, 281–293.
[2] Shatha Alsaedi, Asaad Ahmed, Amin Noaman, and Fathy Eassa. 2024. Two-Level Information-Retrieval-Based Model for Bug Localization Based on Bug Reports. *Electronics* 13 (01 2024), 321.
[3] C. C. Dantas, A. M. Rocha, and M. A. Maia. 2023. How do Developers Improve Code Readability? An Empirical Study of Pull Requests. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE Computer Society, Los Alamitos, CA, USA, 110–122.
[4] Carlos E. Dantas, Adriano M. Rocha, and Marcelo A. Maia. 2023. Assessing the Readability of ChatGPT Code Snippet Recommendations: A Comparative Study. In *Proceedings of the XXXVII Brazilian Symposium on Software Engineering (SBES '23)*. Association for Computing Machinery, New York, NY, USA, 283–292.
[5] DEEPSEEK. 2025. Deep Seek Coder Model. https://deepseekcoder.github.io/. Accessed on June 10, 2025.
[6] F. Deissenbock and M. Pizka. 2005. Concise and consistent naming [software system identifier naming]. In *13th International Workshop on Program Comprehension (IWPC'05)*. 97–106.
[7] GITHUB. 2025. GraphQl V4. https://developer.github.com/v4/. Accessed: 2025-02-20.
[8] Qi Guo, Junming Cao, Xiaofei Xie, Shangqing Liu, Xiaohong Li, Bihuan Chen, and Xin Peng. 2024. Exploring the Potential of ChatGPT in Automated Code Refinement: An Empirical Study. In *Proceedings of the IEEE/ACM 46th* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery.
[9] Julyanara R. Silva and Marcelo A. Maia and Carlos Eduardo C. Dantas. 2025. Do LLMs Suggest Consistent Identifiers? An Empirical Study on GitHub Pull Requests. https://doi.org/10.5281/zenodo.15872389. Accessed: 2025-07-13.
[10] Bin Lin, Simone Scalabrino, Andrea Mocci, Rocco Oliveto, Gabriele Bavota, and Michele Lanza. 2017. Investigating the Use of Code Analysis and NLP to Promote a Consistent Usage of Identifiers. 81–90.
[11] Robert C. Martin. 2008. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, Upper Saddle River, NJ.
[12] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an LLM to Help With Code Understanding. In *Proceedings of the IEEE/ACM 46th* (Lisbon, Portugal) *(ICSE '24)*.
[13] Delano Oliveira, Reydne Santos, Benedito de Oliveira, Martin Monperrus, Fernando Castor, and Fernanda Madeiral. 2025. Understanding Code Understandability Improvements in Code Reviews. *IEEE Transactions on Software Engineering* 51, 1 (2025), 14–37.
[14] Jevgenija Pantiuchina, Fiorella Zampetti, Simone Scalabrino, Valentina Piantadosi, Rocco Oliveto, Gabriele Bavota, and Massimiliano Di Penta. 2020. Why Developers Refactor Source Code: A Mining-based Study. *ACM Trans. Softw. Eng. Methodol.* 29, 4, Article 29 (Sept. 2020), 30 pages.
[15] Nils Reimers. 2023. SentenceTransformer. https://huggingface.co/sentence-transformers. Accessed: 2025-05-11.
[16] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 3982–3992. https://aclanthology.org/D19-1410
[17] Caitlin Sadowski, Kathryn T. Stolee, and Sebastian Elbaum. 2015. How developers search for code: a case study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 191–201.
[18] Julyanara Silva, Carlos Dantas, and Marcelo Maia. 2024. What Developers Ask to ChatGPT in GitHub Pull Requests? an Exploratory Study. In *Anais do XII Workshop de Visualização, Evolução e Manutenção de Software* (Curitiba/PR). SBC, Porto Alegre, RS, Brasil, 125–136.
[19] Dominik Sobania, Martin Briesch, Carol Hanna, and Justyna Petke. 2023. An Analysis of the Automatic Bug Fixing Performance of ChatGPT . In *2023 IEEE/ACM International Workshop on Automated Program Repair (APR)*. IEEE Computer Society, Los Alamitos, CA, USA, 23–30.
[20] SONARLINT. 2025. Sonarlint Web Page. https://www.sonarsource.com/products/sonarlint/. Accessed: 2025-02-20.
[21] Andreas Thies and Christian Roth. 2010. Recommending rename refactorings. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering* (Cape Town, South Africa) *(RSSE '10)*. Association for Computing Machinery, New York, NY, USA, 1–5.
[22] Rosalia Tufano, Antonio Mastropaolo, Federica Pepe, Ozren Dabić, Massimiliano Di Penta, and Gabriele Bavota. 2024. Unveiling ChatGPT's Usage in Open Source Projects: A Mining-based Study.
[23] Zejun Zhang, Zhenchang Xing, Dehai Zhao, Xiwei Xu, Liming Zhu, and Qinghua Lu. 2024. Automated Refactoring of Non-Idiomatic Python Code With Pythonic Idioms. *IEEE Transactions on Software Engineering* PP (11 2024), 1–22.