

# Evaluating ChatGPT's Ability to Detect Naming Bugs in Java Methods

Kinari Nishiura<sup>1</sup><sup>a</sup>, Atsuya Matsutomo<sup>2</sup> and Akito Monden<sup>3</sup><sup>b</sup>

<sup>1</sup>Faculty of Information and Human Sciences, Kyoto Institute of Technology, Kyoto, Japan

<sup>2</sup>School of Engineering, Okayama University, Okayama, Japan

<sup>3</sup>Graduate Scho

Okayama, Japan

*k-nishiura@kit.ac.jp, pqll69dj@s.okayama-u.ac.jp, monden@okayama-u.ac.jp*

**Keywords:** Java, Naming Bug, ChatGPT, Large Language Models, Machine Learning.

**Abstract:** In Java programming, large-scale and complex functions are realized by combining multiple methods. When method names do not match their functionality, readability decreases, making maintenance challenging. Although several machine learning models have been proposed to detect such naming bugs, they require extensive training data, limiting user accessibility. Recently, large language models (LLMs) like ChatGPT have gained popularity and show potential for code comprehension tasks. This study evaluates the performance of ChatGPT in detecting naming bugs using the same datasets as in previous machine learning studies. We evaluated detection accuracy through traditional methods, various prompt adjustments, and more direct approaches. The results indicate that, while ChatGPT does not surpass traditional models, it can match their accuracy with appropriately structured prompts, requiring no additional training.

## 1 INTRODUCTION


In software development, large-scale and complex functions are typically realized by combining multiple processing units. In Java programming specifically, methods and classes represent the smallest units that implement specific functionalities. Developers have the flexibility to name methods and classes freely; however, the names of methods are particularly expected to intuitively reflect their behavior (McConnell, 2004). While method names do not influence function execution, they are crucial for code readability and serve as valuable cues for understanding the code, especially when reviewed by other developers (Boswell and Foucher, 2011; Martin, 2008). When method names fail to align with their behavior, reducing readability, this issue is referred to as a "naming bug" (Høst and Østvold, 2009), and it requires correction.


Several methods for detecting naming bugs have been proposed. Liu et al. (Liu et al., 2019) introduced a technique utilizing Doc2Vec, Word2Vec, and CNN models. Minehisa et al. proposed methods using Doc2Vec and Sent2Vec (Minehisa et al., 2021), as well as approaches based on machine learning models

leveraging Transformer architectures (Minehisa et al., 2022). However, these approaches require collecting appropriate training data and training models, which can pose challenges for easy implementation by users.

Recently, the proliferation of AI services powered by pre-trained large language models (LLMs), such as ChatGPT<sup>1</sup>, has attracted significant attention. These services enable users to interact with AI trained on a vast corpus of web-based knowledge through inputs known as prompts. The notable strengths of pre-trained models include their advanced comprehension and adaptability across various use cases. Leveraging these capabilities, such models can also interpret source code written in programming languages, suggesting the potential to detect naming bugs with prompts specifically designed for code analysis.

In this study, we evaluate ChatGPT's performance in detecting naming bugs using the same dataset employed in previous research focused on machine learning-based detection (Liu et al., 2019; Minehisa et al., 2021; Minehisa et al., 2022). Our experiments target method-level data extracted from 430 open-source software projects, utilizing GPT-3.5 Turbo and GPT-4o mini via a web API. We evaluate not only with the detection criteria established in prior studies but also by modifying the input information and em-

<sup>a</sup> <https://orcid.org/0009-0007-7168-1500>

<sup>b</sup> <https://orcid.org/0000-0003-4295-207X>

<sup>1</sup><https://openai.com/chatgpt>

ploying more direct detection strategies. The results indicate that, while ChatGPT does not significantly outperform existing machine learning models, it can achieve comparable detection performance depending on how the input information is structured. This means that by using ChatGPT, each user can build a machine learning model and rationally save the time and effort required to collect training data and perform training in the task of naming bug detection.

The structure of the paper is as follows. Section 2 reviews the related work referenced in this study. Section 3 details the experimental setup. Section 4 shows the experimental results. Section 5 conducts additional experiments based on the results of the experiment and deepens the discussion. Section 6 discusses threats to validity. Finally, Section 7 concludes the paper.

## 2 RELATED WORK

Several methods utilizing machine learning models have been proposed for detecting naming bugs. Liu et al. (Liu et al., 2019) introduced an approach that combines natural language processing and machine learning techniques to automatically evaluate method names using Doc2Vec, Word2Vec, and CNN. Minehisa et al. (Minehisa et al., 2021) reported that replacing the combination of Word2Vec and CNN with the more lightweight Sent2Vec model could achieve nearly equivalent performance. Additionally, Minehisa et al. (Minehisa et al., 2022) proposed a method where a Transformer-based estimation model predicts method names based on the method's content and verifies whether the predicted names match the actual ones.

In these methods, the detection of naming bugs is not performed as a direct binary classification task to determine whether a method name is appropriate. Instead, the models infer a method name from the method's Abstract Syntax Tree (AST) and assess whether the inferred name's prefix matches the actual method name's prefix. For example, if a method named `getReadTimeout` has its prefix inferred as `get` by the model based on the given AST, the method name `getReadTimeout` is considered free of naming bugs. Conversely, if the model infers a different prefix (e.g., `set` or `write`), `getReadTimeout` would be identified as having a naming bug.

Our experiment partially follows this method of judgement.

## 3 EXPERIMENT DESIGN

### 3.1 Overview

Through evaluation experiments using a dataset containing actual naming bugs, we compared and assessed the accuracy of the proposed method, traditional methods for naming bug detection, and detection using ChatGPT. The comparative methods include three prior approaches: the method by Liu et al. using Doc2Vec, Word2Vec, and CNN (Liu et al., 2019); the approach by Minehisa et al. using Sent2Vec (Minehisa et al., 2021); and the Transformer-based method by Minehisa et al. (Minehisa et al., 2022). For naming bug detection using ChatGPT, in addition to employing the same detection approach as the traditional methods, a total of six different detection patterns were tested.

This experiment utilized the same dataset and evaluation metrics as in previous research (Minehisa et al., 2022). Consequently, the detection accuracies of the traditional methods were taken directly from the reported results of previous research (Minehisa et al., 2022). Details on the frameworks and parameters of these machine learning models can be found in the original papers. The dataset consisted of Java method data collected from 430 open-source software projects. Detailed descriptions of the detection patterns using ChatGPT and the dataset are provided in Sections 3.2 and 3.3, respectively.

Two versions of ChatGPT models were used: GPT-3.5 Turbo and GPT-4o mini. GPT-3.5 Turbo is accessible to general users for free via ChatGPT, while GPT-4o mini, although more advanced than GPT-3.5, offers higher speed and lower costs, making it suitable for this experiment. The experiments were conducted in October 2024, specifically referencing `gpt-3.5-turbo-0125` and `gpt-4o-mini-2024-07-18`. The automation of ChatGPT interactions was achieved using a Python library through the Web API, with the library `openai 1.52.1` used for implementation.

### 3.2 Use of ChatGPT

In this experiment, we evaluate not only the case where the pre-trained model from previous studies is simply replaced with ChatGPT, but also scenarios where the input information to the model is modified or where the evaluation method itself is changed. The six patterns for using ChatGPT to detect naming bugs in this experiment are outlined below, along with abbreviations for each pattern used in the experiments.

**ex1A:** The method content is represented by an AST-analyzed token sequence, and the original

### Prompt

You are a genius mastering the Java language. Please output only a one-line answer to the question below with no introduction, no explanation, no embellishments, only code.

I will show you the flattened token sequence of a Java method's AST with node information, and the method name with the first word hidden.

Please propose a perfect method name that anyone can understand, keeping in mind "what kind of processing is performed" and "what kind of value is returned. Please output only your proposed method name. No reason is required.

[Method name]

\*\*\*ReadTimeout

[AST]

ReturnStatement return ThisExpression this VariableName integerVar

Figure 1: Example of prompt for ex1A.

method name's prefix is concealed. ChatGPT is tasked with suggesting the most appropriate method name, and a naming bug is detected if the suggested name's prefix does not match the original. This method follows the exact evaluation approach used in previous studies.

- ex1J:** In addition to hiding the original method name's prefix, the method content is provided as Java source code instead of an AST-analyzed token sequence. The method name in the source code is concealed.
- ex2A:** Only the AST-analyzed token sequence of the method content is provided, and ChatGPT is tasked with suggesting the appropriate method name, without providing the original method name.
- ex2J:** Only the Java source code of the method content is given, and ChatGPT is tasked with suggesting the appropriate method name. The method name in the source code is concealed.
- ex3A:** Instead of suggesting a method name, both the original method name and the AST-analyzed token sequence of the method content are provided, and ChatGPT is asked to directly assess whether the method name is appropriate.
- ex3J:** Both the original method name and the Java source code of the method content are provided, and ChatGPT is asked to directly assess whether the method name is appropriate.

In this study, the instructions given to ChatGPT are referred to as "prompts." An example of the prompt used in ex1A is shown in Figure 1. This prompt consists of three paragraphs followed by a

Table 1: Variations in the second paragraph of the prompt used in each experiment.

ex	sentences
ex1A	I will show you the flattened token sequence of a Java method's AST with node information, and the method name with the first word hidden.
ex1J	I will show you a Java method source code without method name, and the method name with the first word hidden.
ex2A	I will show you only I will show you the flattened token sequence of a Java method's AST with node information.
ex2J	I will show you a Java method source code. The method name is hidden.
ex3A	I will show you the method name, and the flattened token sequence of a Java method's AST with node information.
ex3J	I will show you the method name, and the Java method source code without method name.

Table 2: Variations in the third paragraph of the prompt used in each experiment.

ex	sentences
ex1, ex2	Please propose a perfect method name that anyone can understand, keeping in mind "what kind of processing is performed" and "what kind of value is returned. Please output only your proposed method name. No reason is required.
ex3	Please determine if the method name is clear enough for everyone to understand, keeping in mind "what kind of processing is performed" and "what kind of value is returned." Please output only "Yes" or "No". No reason is required.

method name and its corresponding Abstract Syntax Tree (AST). The first paragraph specifies the expected behavior from ChatGPT. It has been reported that providing clear and specific roles and instructions in prompts generally leads to better results (Bsharat et al., 2024), which we followed in this study. The second paragraph describes the method information given to ChatGPT, while the third paragraph requests the output from ChatGPT.

The second and third paragraphs of the prompt are modified for each experiment, from ex1A to ex3J, as needed. Variations of the second paragraph are summarized in Table 1, while the variations of the third

paragraph are shown in Table 2. Additionally, the method information provided after the sentences must be appropriately adjusted for each experiment.

### 3.3 Datasets

To evaluate the detection accuracy of the proposed method, we used the same dataset<sup>2</sup> as in Liu et al.'s prior research (Liu et al., 2019). This dataset was also used in the studies by Minehisa et al. (Minehisa et al., 2021)(Minehisa et al., 2022).

The dataset comprises a total of 2,119,218 methods extracted using a proprietary algorithm from 430 open-source Java projects. In previous studies, 2,116,413 of these methods were used as training data, while 2,805 were designated as test data. For our experiment, since we leveraged ChatGPT, which has already been pre-trained on extensive web-based data, we only used the test data for evaluation. Out of this test data, only 1,268 methods could be matched correctly with their method names and corresponding Java source code from the published dataset. Among these, 635 were labeled as containing naming bugs, and 633 were labeled as bug-free.

The method for constructing the dataset as described in Liu et al.'s study (Liu et al., 2019) is outlined below. First, all methods from the 430 open-source projects were collected, excluding: (1) Main methods, (2) Constructors, (3) Empty methods (abstract methods or methods with no operations), (4) Methods with names containing "example," "sample," or "template," (5) Methods whose names do not include alphabetic characters (e.g., methods using only underscores).

Methods with more than 95 tokens after AST transformation were excluded due to model limitations, resulting in the final dataset of 2,119,218 methods. Subsequently, GitHub commit histories of the projects were analyzed to identify methods meeting the following criteria as candidates for naming bugs:

**Criterion 1:**

The commit involved a change where only the method name was modified, without altering the method's body.

**Criterion 2:**

The reason for the name change was not to correct a typographical error.

**Criterion 3:**

The method's content and name remained unchanged after the renaming commit until the time of data collection.

Methods meeting Criterion 1 were expected to represent cases where a suboptimal name was corrected to an appropriate one. Criterion 2 aimed to remove noise, while Criterion 3 ensured the consistency of the data. This process yielded 2,805 methods as test data. Commit history analysis provided the names before and after the renaming, and visual inspection by the authors of the original study (Liu et al., 2019) confirmed whether the initial names constituted naming bugs. As a result, 1,403 methods were labeled as containing naming bugs, while 1,402 were labeled as bug-free.

Next, we describe the matching process between method names and Java source code. While the published dataset maintained sequential order for the original and revised method names, tokenized ASTs, and labels, it did not do so for the Java source code. Given that the source code contained the updated method names, we performed a straightforward matching procedure, restricting the linkage to methods with unique names within the dataset. This approach resulted in 1,268 matched test data points, which were used in our experiment.

### 3.4 Evaluation Metrics

Following prior research, the evaluation metrics for naming bug detection include accuracy, precision, recall, and F1-score. The formulas for these metrics are shown in Table 3. The confusion matrix used for deriving these metrics is presented in Table 4.

The criteria for determining naming bugs vary depending on the specific approach used with ChatGPT. When employing the method similar to previous studies, where an appropriate method name is predicted and evaluated based on prefix matching, the comparison baseline depends on the labels in the test data. For test data labeled as containing naming bugs, the predicted method name is compared to the pre-correction name. For data labeled as bug-free, the comparison is made with the post-correction name. This approach aligns with the methodology of existing research for consistency in evaluation. In the method where ChatGPT directly judges the presence of naming bugs, the results provided by ChatGPT are used without further transformation.

Given that ChatGPT's responses can exhibit variability, its heuristic nature must be considered. To mitigate this variability, naming bug detection for each method was conducted independently three times, with the majority outcome being taken as the final result.

<sup>2</sup><https://github.com/TruX-DTF/debug-method-name/tree/master/Data/TestData>

Table 3: Evaluation metrics formula.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad (1)$$

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

$$F1 = \frac{2 \times recall \times precision}{recall + precision} \quad (4)$$

Table 4: Confusion matrix.

	Judged as naming bug	Judged as not naming bug
Naming bug	True Positive TP	False Negative FN
Not naming bug	False Positive FP	True Negative TN

## 4 EXPERIMENT RESULTS

The results of the experiment are shown in Table 5, with the best results in each metric highlighted in bold. The highest scores were observed as follows: accuracy and precision with GPT-4o mini in ex2J, recall with GPT-3.5 Turbo in ex1A, and F1-score with GPT-3.5 Turbo in ex2A.

Firstly, there was minimal difference in F1-scores between GPT-3.5 Turbo and GPT-4o mini for most of the ChatGPT usage methods. The exception is when determining naming bugs directly from given method name and AST token sequences, where GPT-3.5 Turbo showed a significantly lower performance. This was primarily due to GPT-3.5 Turbo in this case often predicting almost all cases as bug-free, resulting in very low recall. In contrast, GPT-4o mini exhibited consistent performance whether given AST tokens or Java source code, suggesting greater robustness.

Next, the method of directly asking ChatGPT to detect naming bugs (ex3) yielded a lower F1-score compared to the method of generating a suggested method name and comparing its prefix to the actual name. This indicates that using ChatGPT for naming bug detection is more effective when following the methodology of previous research, where an appropriate method name is proposed based on the method content and compared for prefix differences.

When comparing the information provided with ChatGPT, there was little difference between using AST token sequences and Java source code (except for ex3). Given the ease of providing raw Java source code, this approach is more practical. Additionally, there was no significant difference between providing the actual method name with the hidden prefix (ex1) and not providing any method name information (ex2). This result implies that given partial method names do not introduce noise, but also hints related to method names are not particularly necessary.

In summary, using ChatGPT to directly judge naming bugs is less effective. Using GPT-3.5 Turbo for ex1A (providing AST token sequences and a hidden prefix) slightly lowered the F1-score compared to other cases, but overall, most approaches yielded similar performances.

Next, we will compare the accuracy of the proposed model with that of previous studies. The highest accuracy reported was by Minehisa et al.'s Transformer-based approach among previous studies, with an F1-score of 0.679 (Minehisa et al., 2022). The direct judgment method (ex3) performed worse than this, while the prefix comparison methods (ex1 and ex2) occasionally surpassed this score but did not show significant overall differences. ChatGPT outperformed non-Transformer prior methods, indicating that while it may not exceed the most advanced models without fine-tuning, it can achieve comparable results with straightforward data input.

This suggests that using ChatGPT for naming bug detection offers substantial convenience, as it does not require specialized training, skillsets, or extensive data collection by each user. Ordinary programmers can utilize ChatGPT for effective naming bug detection without deep expertise.

Our experiment showed that for each method, three independent trials were conducted, with the majority result used as the final outcome. An analysis of the consistency across trials, shown in Table 6, revealed that GPT-4o mini exhibited higher consistency compared to GPT-3.5 Turbo. Consistency was lowest for ex3, while ex1 and ex2 with GPT-4o mini had approximately 90% consistency. This indicates that while ChatGPT responses have inherent randomness, GPT-4o mini used with prefix comparison shows strong result consistency, reducing the need for multiple trials.

Table 5: Experiment results.

		Accuracy	Precision	Recall	F1-score
Doc2Vec, Word2Vec, and CNN(Liu et al., 2019)		0.482	0.489	0.775	0.599
Sent2Vec(Minehisa et al., 2021)		0.531	0.519	0.843	0.643
Transformer(Minehisa et al., 2022)		0.582	0.551	0.884	0.679
ex1A	3.5 Turbo	0.506	0.502	<b>0.967</b>	0.661
	4o mini	0.584	0.553	0.863	0.674
ex1J	3.5 Turbo	0.599	0.563	0.878	0.686
	4o mini	0.605	0.574	0.809	0.671
ex2A	3.5 Turbo	0.573	0.541	0.940	<b>0.687</b>
	4o mini	0.603	0.567	0.869	0.686
ex2J	3.5 Turbo	0.618	<b>0.582</b>	0.826	0.683
	4o mini	<b>0.619</b>	0.584	0.821	0.683
ex3A	3.5 Turbo	0.498	0.492	0.152	0.232
	4o mini	0.509	0.507	0.594	0.547
ex3J	3.5 Turbo	0.501	0.500	0.423	0.459
	4o mini	0.512	0.509	0.646	0.569

Table 6: The percentage of times ChatGPT gave the same answer all three times.

	GPT 3.5 Turbo	GPT 4o mini
ex1A	90.9%	92.4%
ex1J	88.2%	94.0%
ex2A	89.4%	89.7%
ex2J	93.3%	95.2%
ex3A	64.6%	80.4%
ex3J	63.6%	85.3%

Table 7: Naming bug detection accuracy by ChatGPT 4o mini for methods with verb-based prefixes.

	Accuracy	Precision	Recall	F1-score
ex1A	0.544	0.460	0.726	0.563
ex1J	0.586	0.491	0.634	0.553
ex2A	0.574	0.483	0.743	0.585
ex2J	0.603	0.508	0.653	0.571
ex3A	0.451	0.364	0.479	0.414
ex3J	0.436	0.364	0.528	0.431

## 5 ADDITIONAL EXPERIMENTS AND DISCUSSIONS

This section explores additional analyses and experiments conducted following the previous experimental results.

### 5.1 Detection Accuracy Evaluation Limited to Methods with Verb Prefixes

A manual inspection of the method names suggested by ChatGPT revealed that nearly all of them had prefixes that were verbs. This aligns with the common best practice that method names should start with verbs. However, upon reviewing the original method names in the dataset, there was a noticeable number of cases where the prefixes were not verbs. This discrepancy in prefix type between the suggested and original method names could potentially lower the detection

accuracy for ex1 and ex2. Therefore, we evaluated the accuracy specifically for methods whose original prefixes were verbs.

To identify the part of speech, we used the natural language processing library spaCy<sup>3</sup>. We filtered out methods whose prefixes were not verbs (“VERB”) or auxiliary verbs (“AUX”). This left us with 748 methods, of which 303 were labeled as containing naming bugs and 445 were labeled as bug-free.

The naming bug detection accuracy only for these methods, using GPT-4o mini, is shown in Table 7. Compared to Table 5, contrary to our expectations, indicating that restricting the evaluation to methods with verb prefixes led to a decline in detection performance. The main reason appears to be that, for methods with verb prefixes, ChatGPT’s predictions were split more evenly between positive (naming bug present) and negative (no naming bug) outcomes. In contrast, when evaluating other methods, most were predicted as positive, leading to very high recall and consequently higher F1-scores. This analysis suggests that even for ChatGPT, detecting naming bugs

<sup>3</sup><https://spacy.io>

Table 8: Naming bug detection accuracy using semantic similarity of prefixes by ChatGPT 4o mini.

	Accuracy	Precision	Recall	F1-score
ex1A	0.606	0.580	0.760	0.658
ex1J	0.602	0.588	0.673	0.628
ex2A	0.615	0.588	0.765	0.665
ex2J	0.614	0.598	0.695	0.643

in method names with verb prefixes is more challenging.

## 5.2 Evaluation of Detection Accuracy Based on Semantic Similarity of Prefixes

In the experiments described in Section 3, ex1 and ex2 used exact matches of method name prefixes to determine naming bugs. However, due to the existence of synonyms in English, prefixes that do not match exactly might still be semantically similar, potentially leading to false positives in naming bug detection. Examples of such synonymous prefixes include *get* and *obtain*, *start* and *begin*, or *run* and *execute*.

To address this, we evaluated detection accuracy considering semantic similarity in prefixes. While several natural language processing techniques can be employed for synonym detection, we chose a straightforward approach using ChatGPT for this task. Specifically, we used the GPT 4o mini model and performed a single judgment for each comparison.

The results of using semantic synonym detection for ex1 and ex2 (where ChatGPT 4o mini was used to propose method names) are shown in Table 8. Compared to Table 5, precision slightly increased, but recall and F1-score decreased slightly, resulting in a detection performance significantly lower than that of Minehisa et al.’s method.

This suggests that considering semantic similarity when detecting naming bugs using prefix matching does not improve accuracy. One possible explanation is that ChatGPT’s synonym detection may not be entirely reliable, which calls for more robust synonym detection techniques in future research.

## 5.3 Evaluation of Detection Accuracy with Stricter Criteria for ChatGPT

In the experiments of Section 3, ex3 involved asking ChatGPT whether a method name was “clear enough for everyone to understand,” and naming bugs were identified based on negative responses. While this

Table 9: Naming bug detection accuracy of ChatGPT 4o mini with modified prompt criteria.

	Accuracy	Precision	Recall	F1-score
ex3A	0.539	0.647	0.171	0.270
ex3J	0.735	0.874	0.548	0.674

criterion was somewhat suitable, it included a subjective element that could make the guidance ambiguous, particularly when assessing edge cases or “middle layer.”

To address this, we refined the prompt to enforce stricter criteria for determining naming bugs, focusing on detecting method names that were clearly inappropriate. For this purpose, the third sentence of the prompts for ex3A and ex3J was modified as follows: “Please judge whether the method name given is clearly inappropriate. Please output only ‘Yes’ or ‘No’ in one line. No reason is required.” This was conducted using the GPT 4o mini model with a single assessment per method.

The results are shown in Table 9. The data reveals a significant decrease in accuracy when AST token sequences were provided, whereas the accuracy improved when Java source code was used as input. Compared to the method by Minehisa et al., the F1-score became comparable, and accuracy and precision were considerably higher.

This indicates that, although prior to the prompt modification, ex3 showed significantly lower accuracy than the existing method, adjusting the prompt to detect only clearly inappropriate names achieved comparable accuracy to established approaches. This suggests that users can rely on ChatGPT for naming bug detection without performing indirect and labor-intensive analyses like proposing method names and comparing prefixes, enhancing convenience.

One possible reason for the improvement in accuracy is that ChatGPT’s judgement for the ‘middle layer’, which is not completely appropriate but also not completely inappropriate, did not match the labelling of the authors of the previous study. They judged that many of the ‘middle layer’ were not naming bugs, and the improved prompts also provide the same judgement criteria.

However, recall was still much lower than that of existing methods, implying that if comprehensive detection is a priority, existing approaches remain more suitable. The reason for the drastic drop in accuracy when AST token sequences were used remains unclear and may warrant further investigation.

## 6 THREATS TO VALIDITY

Potential threats to the validity of this study include the following:

First, we compared our results with the detection accuracy reported in previous work (Minehisa et al., 2022) without conducting replication experiments of prior methods due to time constraints. While the previous research evaluated a total of 2,805 test data points, our study was limited to 1,268 test cases that could be aligned with corresponding source code. This discrepancy in the evaluation dataset could potentially affect the detection accuracy. Future work should include replicating the prior methods and evaluating them using the exact same dataset to ensure a fair comparison.

Second, the test data used in this study were labeled by the authors of prior research (Liu et al., 2019) to indicate the presence or absence of naming bugs. The accuracy of these labels cannot be guaranteed. Additionally, the criteria for determining naming bugs that we provided to ChatGPT may differ from those used in the original labeling. Re-labeling the data ourselves and aligning the labeling criteria with those used for ChatGPT would enable a more equitable comparison.

Lastly, due to time and resource constraints, only GPT 3.5 Turbo and GPT 4o mini were used as ChatGPT models. Higher-performing models, such as more advanced versions of GPT-4, may yield improved results. Exploring other generative AI services could also be beneficial. However, given the simplicity of the tasks assigned in this study, it is possible that the use of more advanced models would not significantly affect the accuracy.

## 7 CONCLUSION

In this study, we used ChatGPT to detect naming bugs and compared the detection accuracy with that of conventional methods. In the experiment, the same evaluation data as in previous research was used, and in addition to the same detection method as in the conventional method, we also evaluated cases where the input was changed and cases where direct binary classification was performed. In addition, both the 3.5 Turbo and the 4o mini ChatGPT models were used. As a result, it was found that ChatGPT does not greatly exceed the detection performance of existing machine learning models, but that it has detection performance equivalent to existing methods without special training, depending on how the information is given. There was no difference in the ChatGPT mod-

els used, and there was also no significant difference between the cases where the token sequences were given to ChatGPT after AST analysis, as in the previous study, and the cases where the Java source code was given as is. When ChatGPT was given method names and processing content to directly detect naming bugs, it was found to perform at the same level as previous research when given Java source code and prompted to detect bugs using stricter criteria.

As a result, using ChatGPT to detect naming bugs is effective in increasing developer convenience, and it is possible that naming bugs can be pointed out and fixed more easily.

Future issues include more detailed evaluation of existing methods and expanding the naming bug data used for evaluation. More specifically, comparisons with recent related research (Wang et al., 2024) that was not included in this experiment, and comparisons with other methods that focus on the generation of method names, are important issues for the future. It is also possible to consider evaluating other LLMs or expanding the evaluation to other programming languages besides Java.

## ACKNOWLEDGEMENTS

This work was supported in part by JSPS KAKENHI Grant number JP23K16863 and JP20H05706.

## REFERENCES

- Boswell, D. and Foucher, T. (2011). *The Art of Readable Code: Simple and Practical Techniques for Writing Better Code*. O'Reilly & Associates.
- Bsharat, S. M., Myrzakhan, A., and Shen, Z. (2024). Principled instructions are all you need for questioning llama-1/2, gpt-3.5/4.
- Høst, E. W. and Østvold, B. M. (2009). Debugging method names. In *Proceedings of the 23rd European Conference on ECOOP 2009 — Object-Oriented Programming*, Genoa, page 294–317, Berlin, Heidelberg. Springer-Verlag.
- Liu, K., Kim, D., Bissyandé, T., Taeyoung, K., Kim, K., Koyuncu, A., Kim, S., and Le Traon, Y. (2019). Learning to spot and refactor inconsistent method names. In *in Proc. 41st Int'l Conf. Softw. Eng.*, pages 1–12.
- Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.
- McConnell, S. (2004). *Code Complete, Second Edition*. Microsoft Press, USA.
- Minehisa, T., Aman, H., and Kawahara, M. (2022). Naming bug detection using transformer-based method name



suggestion. *Computer Software (in Japanese)*, pages 17–23.

Minehisa, T., Aman, H., Yokogawa, T., and Kawahara, M. (2021). A comparative study of vectorization approaches for detecting inconsistent method names. *Computer and Information Science 2021—Summer*, 985:125–144.

Wang, T., Zhang, Y., Jiang, L., Tang, Y., Li, G., and Liu, H. (2024). Deep learning based identification of inconsistent method names: How far are we? *Empirical Software Engineering*, 30.

