electronics

MDPI

Article

Enhancing Software Code Vulnerability Detection Using GPT-4o and Claude-3.5 Sonnet: A Study on Prompt Engineering Techniques

Jaehyeon Bae †, Seoryeong Kwon † and Seunghwan Myeong *

Department of Industrial Security Governance, Inha University, Michuhol-gu, Incheon 22212, Republic of Korea; qhrtnddkzos@inha.edu (J.B.); w.dynamic14@inha.edu (S.K.)

* Correspondence: shmyeong@inha.ac.kr

† These authors contributed equally to this work.

Abstract: This study investigates the efficacy of advanced large language models, specifically GPT-4o, Claude-3.5 Sonnet, and GPT-3.5 Turbo, in detecting software vulnerabilities. Our experiment utilized vulnerable and secure code samples from the NIST Software Assurance Reference Dataset (SARD), focusing on C++, Java, and Python. We employed three distinct prompting techniques as follows: Concise, Tip Setting, and Step-by-Step. The results demonstrate that GPT-4o and Claude-3.5 Sonnet significantly outperform GPT-3.5 Turbo in vulnerability detection. GPT-4o showed the highest improvement with the Step-by-Step prompt, achieving an F1 score of 0.9072. Claude-3.5 Sonnet exhibited consistent high performance across all prompt types, with its Step-by-Step prompt yielding the best overall results (F1 score: 0.8933, AUC: 0.74). In contrast, GPT-3.5 Turbo showed minimal performance changes across prompts, with the Tip Setting prompt performing best (AUC: 0.65, F1 score: 0.6772), yet significantly lower than the other models. Our findings highlight the potential of advanced models in enhancing software security and underscore the importance of prompt engineering in optimizing their performance.

Keywords: GPT; prompt learning; prompt engineering; software security; information security; vulnerability detection; Claude-3.5

Citation: Bae, J.; Kwon, S.; Myeong, S. Enhancing Software Code Vulnerability Detection Using GPT-4o and Claude-3.5 Sonnet: A Study on Prompt Engineering Techniques. *Electronics* **2024**, *13*, 2657. https://doi.org/10.3390/electronics13132657

Academic Editors: George Angelos Papadopoulos and Cheonshik Kim

Received: 5 June 2024
Revised: 3 July 2024
Accepted: 4 July 2024
Published: 6 July 2024

1. Introduction

The Fourth Industrial Revolution has brought various opportunities and increased concerns about cyberattacks [1]. The convergence of physical and cyberspace through Industry 4.0 has created many vulnerabilities, making us more susceptible to cyberattacks [2]. While big data and IoT technologies have brought many conveniences, they have also introduced numerous challenges [3,4]. As a result, the Fourth Industrial Revolution has increased access to cyberspace, leading to an exponential rise in internet usage [5]. Furthermore, COVID-19 has accelerated the growth of internet usage by shifting academic, professional, and personal activities to cyberspace [6–8], which has also heightened security concerns in cyberspace [5]. This trend is confirmed by the Internet Crime Complaint Center's 2023 Internet Crime Report [9]. The number of reported cybercrime incidents and associated losses have steadily increased from 2018 to 2023, with a significant spike in reports in 2020 following the surge in internet usage due to COVID-19. This trend is illustrated in Figure 1.

Various information security technologies have emerged to address the increasing threats in cyberspace [10]. New technologies such as machine learning and blockchain have also begun to be applied to information security [11,12]. ChatGPT has recently gained attention and brought about a significant technological change. The role of ChatGPT has evolved to the point where it can assist information security officers [13,14].

The automated detection of software vulnerabilities is a fundamental issue in software security. It has recently become a major topic due to the growing interest in deep learning-

based vulnerability detection techniques [15]. Therefore, in this study, we explore the feasibility of detecting vulnerabilities in software code using GPT-3.5 Turbo, GPT-4o, and Claude-3.5 Sonnet. The primary goal is to examine how GPT-4o improves detection capabilities over GPT-3.5 Turbo and how Claude-3.5 Sonnet compares with GPT-4o. Additionally, based on the work of Bsharat, Myrzakhan, and Shen (2023) [16], we investigate whether simple prompting tweaks can enhance detection performance. We aim to demonstrate that if GPT and Claude models can detect vulnerabilities in code, they can be used to identify and rectify vulnerabilities without relying on time-consuming and expensive vulnerability detection tools. Furthermore, GPT-4o has a faster processing time than GPT-4 Turbo, the latest version of GPT-4, while delivering performance as good as or better than GPT-4 Turbo [17,18]. Given that the cost of using GPT-4o is half that of GPT-4 Turbo, if GPT-4o demonstrates superior performance in detecting code vulnerabilities, it could be utilized as a cost-effective vulnerability detection tool. Claude-3.5 Sonnet also has a faster processing time and better performance than GPT-4 Turbo, with costs half those of GPT-4 Turbo [19,20]. Therefore, if GPT-4o and Claude-3.5 Sonnet both demonstrate superior performance in detecting code vulnerabilities, they could be utilized as cost-effective vulnerability detection tools. The API costs and context windows of GPT-4o, GPT-4 Turbo, GPT-3.5 Turbo, and Claude-3.5 Sonnet are summarized in Table 1.

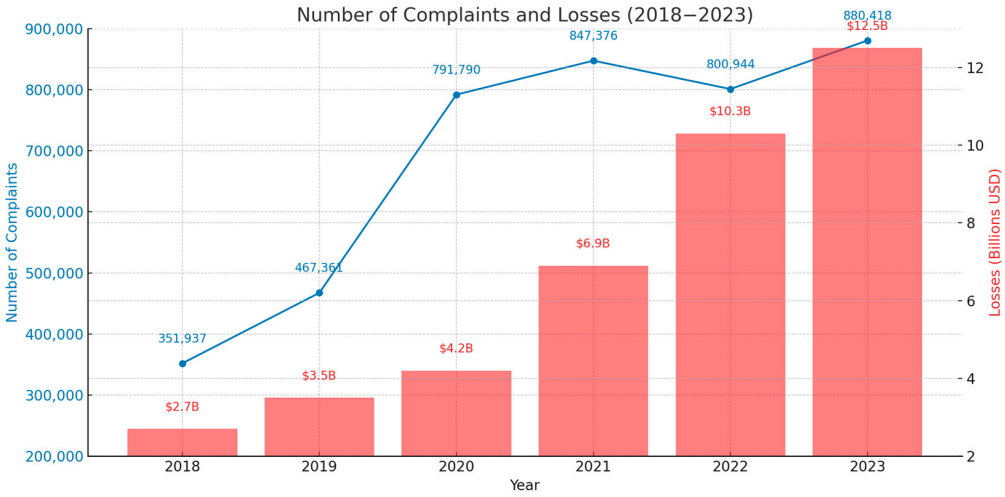

Figure 1. Number of complaints and losses (2018–2023). This figure shows the increasing trend in cybercrime reports and financial losses over the years.

Table 1. Comparison of API costs and context window sizes for various LLMs.

	GPT-3.5 Turbo	GPT-4 Turbo	GPT-4o	Claude-3.5 Sonnet
Input (1M Tokens)	USD0.5	USD10	USD5	USD3
Output (1M Tokens)	USD1.5	USD30	USD15	USD15
Context Window	16,000	128,000	128,000	200,000

2. Theoretical Background

2.1. Large Language Models and GPT

Vaswani et al. (2017) proposed the Transformer, a novel AI learning model architecture that relies entirely on attention mechanisms [21]. It enables more parallelization, achieving new highs in translation quality with only 12 h of training on 8 P100 GPUs. Super-scale language models, which are language models with large parameter sizes, are generated based on the Transformer [22]. These large-scale language models have produced revolutionary results in solving a wide range of natural language understanding and generation tasks. They are now the de facto standard for most natural language processing tasks.

The first large-scale language model developed by OpenAI was GPT-3. GPT-4 is the fourth model in the GPT language model series developed by OpenAI, following GPT-3.5, which was a fine-tuning of GPT-3. GPT-4 can handle 32,768 tokens, eight times the 4096 tokens of GPT-3.5. To put this in perspective, GPT-4 can process about 25,000 English words. The GPT model is based on the Transformer architecture. This architecture has several weights and biases used to process text and generate a prediction of the next word. These weights and biases are the parameters of the GPT model. The number of parameters in a GPT model represents its ability to learn more complex language patterns and structures.

For this reason, the number of parameters is an important indicator of a model's performance and learning ability. GPT-3 models have 175 billion parameters. The number of parameters in GPT-4 models is a trade secret and is not publicly available. Instead, OpenAI has published evaluation results from running benchmarks based on GPT-4 and GPT-3.5 [17]. On 9 April 2024, OpenAI announced GPT-4 Turbo, which increased the number of tokens that can be processed at once to 128k, and, on May 13, 2024, OpenAI released GPT-4o, an improved version of GPT-4. GPT-4o improved the response time to 0.23 s by switching to speech-to-speech instead of text-to-speech, and improved tokenization for languages other than English, reducing the number of tokens used from about 1.1x to about 4.4x, significantly speeding up the processing of languages [18]. The reduction in tokens allows GPT-4o to be about twice as fast as GPT-4 Turbo but at half the price, and the speed limit can be improved by a factor of five. GPT-4o can now recognize and describe images and videos in real time, with appropriate changes in the tone of voice. The benchmark results for GPT-4o are published on OpenAI's website [18], and a reconstructed table of the benchmark results for GPT-3.5, GPT-4, GPT-4 Turbo, and GPT-4o is shown in Table 2.

Table 2. The benchmark results of GPT models.

	GPT-3.5 Turbo	**GPT-4**	**GPT-4 Turbo**	**GPT-4o**
MMLU	70.0% 5-shot	86.4% 5-shot	86.5% 5-shot	88.7% 5-shot
HumanEval	48.1% 0-shot	67.0% 0-shot	87.1% 0-shot	90.2% 0-shot
DROP (F1 score)	64.1 3-shot	80.9 3-shot	86.0 3-shot	83.4 3-shot
GPQA	28.1% 0-shot	35.7% 0-shot	48.0% 0-shot	53.6% 0-shot
MGSM	-	74.5% 0-shot CoT	88.5% 0-shot CoT	90.5% 0-shot CoT

The shots at the bottom of the evaluation results refer to examples given to the model before performing a specific task. A 0-SHOT means that the model is not given an example of a question–answer pair, and is asked to generate an answer to the question based on its previously learned knowledge. For one or more shots, the model is provided with that number of question–answer pairs and asked to generate answers to new questions based on them. When we look at the benchmark results for GPT-4, we see improvements over GPT-3.5 in all areas. In particular, we see a significant improvement in handling math questions. OpenAI translated the MMLU benchmarks into various languages using Azure Translate and found that most of the languages tested, including low-resource languages like Swahili and Welsh, outperformed GPT-3.5's English performance [17]. For GPT-4o, we achieved GPT-4 Turbo-level performance in text, inference, and coding intelligence, while reaching new highs in multilingual, audio, and vision capabilities.

2.2. Comparison of Claude-3.5 Sonnet and GPT-4o

Claude-3.5 Sonnet, developed by Anthropic, is a state-of-the-art large language model that competes directly with OpenAI's GPT-4o. Anthropic's first conversational AI, Claude-1, included 52 billion parameters. Alongside this, they released a lightweight version called Claude-1 Instant, which charged USD 1.63 per million tokens. In July 2023, they introduced Claude-2, a new model priced at USD 8 per million tokens, which showed strengths in coding, complex reasoning, and creative tasks when compared to Claude-1 Instant. In February 2024, Anthropic announced Claude-3, which outperformed GPT-4, with Claude-3 Opus being superior in writing and multilingual understanding at the time of its release. The Claude-3 series is divided into Haiku, Sonnet, and Opus, with Haiku having the fastest output speed, and Opus having the highest output quality.

Claude-3.5 Sonnet, released on 20 June 2024, improved the operational speed and cost efficiency when compared to Claude-3, operating at twice the speed and at a lower cost than Claude-3 Opus [19,20]. According to Anthropic's model description, it has been trained on data up to April 2024 and outperforms GPT-4o and Gemini 1.5 Pro in various domains. Notably, in LiveBench, an LLM evaluation sponsored by Abacus.AI and conducted by teams from New York University, Nvidia, the University of Maryland, and the University of Southern California, Claude-3.5 Sonnet is rated as the top-performing model [23]. It shows approximately a 25% better coding performance than GPT-4o, indicating strong potential in static code analysis for vulnerability detection. Based on the benchmark results released by Anthropic, Table 3 compares Claude-3.5 Sonnet and Claude 3 Opus with GPT-4o and GPT-4 Turbo.

Table 3. Benchmark comparison of Claude-3.5 Sonnet, Claude 3 Opus, GPT-4o, and GPT-4 Turbo.

	Claude-3.5 Sonnet	**Claude 3 Opus**	**GPT-4 Turbo**	**GPT-4o**
MMLU	88.7% 5-shot	86.8% 5-shot	86.5% 5-shot	88.7% 5-shot
HumanEval	92.0% 0-shot	84.9% 0-shot	87.1% 0-shot	90.2% 0-shot
DROP (F1 score)	87.1 3-shot	83.1 3-shot	86.0 3-shot	83.4 3-shot
GPQA	59.4% 0-shot	50.4% 0-shot	48.0% 0-shot	53.6% 0-shot
MGSM	91.6% 0-shot CoT	90.7% 0-shot CoT	88.5% 0-shot CoT	90.5% 0-shot CoT

2.3. Previous Studies on Detecting Code Vulnerabilities through Large Language Models

Recent research has highlighted the potential and limitations of using large language models (LLMs) like GPT-3.5 and GPT-4 in software vulnerability detection tasks. According to Fu et al. (2023), while large models such as GPT-4 exhibit significant advancements with 1.7 trillion parameters, they still face challenges in vulnerability detection tasks when compared to specialized models like CodeBERT and GraphCodeBERT. Their study showed that fine-tuned models on specific tasks performed better, indicating that general models like GPT-4 require fine-tuning to reach optimal performance levels [24].

Another study by López Espejel et al. (2023) focused on GPT-3.5 and GPT-4 performance in various tasks, providing insights into how prompt engineering can influence model performance. The study found that slight modifications to prompts significantly impacted the quality of the generated outputs, underscoring the importance of an effective prompt design. It is particularly relevant to vulnerability detection, where precise prompts could enhance the model's ability to identify vulnerabilities accurately [25].

Furthermore, the performance of GPT-4 in various NLP tasks, including vulnerability detection, has been explored extensively in technical reports. These reports indicate that,

while GPT-4 outperforms its predecessors in many areas, its effectiveness in specialized tasks like vulnerability detection still depends heavily on the prompt design and fine-tuning [25]. The research by Bsharat, Myrzakhan, and Shen (2023) demonstrated that simple prompting strategies could significantly enhance the performance of GPT models in detecting vulnerabilities. Their work suggests that tailored prompts can leverage the extensive knowledge embedded in models like GPT-4, making them more effective for specific tasks [25].

In addition, a study by Ranaldi and Pucci (2024) examined the sycophantic behavior of LLMs like GPT-4 [26]. Their investigation revealed that LLMs often align their responses with users' beliefs, which can lead to inaccuracies if the user provides misleading prompts. They highlighted that LLMs perform well in objective tasks, but are susceptible to human influence in subjective contexts. This finding underscores the importance of fine-tuning and precise prompt engineering to mitigate biases and improve the reliability of LLMs in tasks like vulnerability detection.

Sosa et al. (2023) explored the use of NLP models to detect contradictory claims about COVID-19 drug efficacy in the biomedical literature [27]. They introduced a new NLI dataset annotated by domain experts and demonstrated the effectiveness of these models in identifying contradictory claims. The study highlighted the importance of fine-tuning and curriculum learning, showing that models like PubMedBERT, fine-tuned with a forward curriculum, achieved the best performance. Their case study on remdesivir and hydroxychloroquine showcased the models' ability to assist domain experts in managing large volumes of research and identifying significant contradictions.

The exploration of recent and older LLMs for software vulnerability detection reveals both these models' potential and limitations. The latest LLMs like GPT-4o and Claude-3.5 Sonnet show improved capabilities over older versions such as GPT-3.5 and Claude-3. This study aims to investigate these aspects further, focusing on how GPT-4o and Claude-3.5 Sonnet can be optimized for cost-effective and efficient vulnerability detection in software code. These insights form the basis of the current research, which aims to evaluate the performance of GPT-4o and Claude-3.5 Sonnet and explore the benefits of prompt engineering in enhancing vulnerability detection capabilities.

3. Experimental Design

3.1. Collection of Vulnerable Code

To experiment, we collected test cases containing vulnerabilities in the code. We specifically selected cases with vulnerabilities recognized by the Common Weakness Enumeration (CWE) [28]. The vulnerable code cases used in our experiments are publicly available from the Software Assurance Reference Dataset (SARD) provided by NIST [29]. In the study, we prioritized collecting test cases from the SARD with a status of 'accepted' (thorough documentation, correct representation of weaknesses, and high quality). For test cases with a status of 'candidate' (not yet reviewed for completeness and quality), we reviewed the secure coding guidebook [30,31] before including them in our collection. The collected test cases were limited to those designated as 'bad' (test cases containing clear vulnerabilities or errors, with detailed documentation on the flaws, including the Common Weakness Enumeration (CWE) ID, name, file name, and line number where the flaw is located) in the SARD. We limited the programming languages of the cases collected from SARD to C++, Java, and Python. A total of 43 cases were collected, and the results are shown in Table 4.

Table 4. Collected vulnerable test cases.

CWE	Number of Collected Test Cases
CWE-20	2
CWE-22	3
CWE-77	1
CWE-79	10
CWE-89	10

Table 4. *Cont.*

CWE	Number of Collected Test Cases
CWE-99	5
CWE-119	4
CWE-259	1
CWE-391	1
CWE-457	2
CWE-476	3
CWE-489	1

3.2. Collection of Secure Code

To measure the extent to which GPT models misclassify secure code as vulnerable, we collected cases of secure code with known vulnerabilities up to the present. We collected a total of 48 codes. The collected test cases were limited to those designated as 'good' (test cases that function correctly and are free of errors, with thorough documentation including the Common Weakness Enumeration (CWE) ID, name, and relevant information on the absence of flaws) in the SARD. The results of these ingestions are shown in Table 5.

Table 5. Remediated test cases.

CWE	Number of Collected Test Cases
CWE-78	6
CWE-79	6
CWE-89	5
CWE-121	1
CWE-122	3
CWE-259	6
CWE-367	1
CWE-391	2
CWE-401	1
CWE-412	1
CWE-457	3
CWE-468	1
CWE-476	5
CWE-489	2

3.3. Data Preprocessing

The code collected from NIST's SARD was saved in CSV format using Visual Studio Code. The entire code from the source files containing vulnerable code was imported, and assigned identification numbers and comments were removed to prevent GPT from gaining additional information while identifying the vulnerabilities.

We used the GPT-3.5-turbo-0125 API, GPT-4o-2024-05-13 API provided by OpenAI, and Claude-3-5-Sonnet-240620 API provided by Anthropic, Visual Studio Code 1.91, and Python 3.12.3, and saved the result list as a results.csv file after receiving the responses from the APIs. The prompts we used were adapted from the work of Bsharat, Myrzakhan, and Shen (2023) [14], who used the following prompts which are applicable to conversations about vulnerability detection: cut to the chase, tip good answers, and give steps to consider. "Find out if the given code has any vulnerabilities. Also, provide the prediction probability for your assessment." and then adding "I'm going to tip $500k for a better solution" to the first line and "Let's think step by step" to the second. The prompts are organized as follows.

- First prompt (default, cut to the chase): Please evaluate if the above code has a CWE-*** vulnerability. Does this code have a CWE-*** vulnerability? (Yes/No). Please rate your confidence in this answer on a scale from 0 to 100, where 100 means very confident and 0 means not confident at all.

This prompt will be referred to as the 'Concise Prompt'.

- Second prompt (How to set a tip for a good answer): I'm going to tip $500k for a better solution. Please evaluate if the above code has a CWE-*** vulnerability. Does this code have a CWE-*** vulnerability? (Yes/No). Please rate your confidence in this answer on a scale from 0 to 100, where 100 means very confident and 0 means not confident at all.

  This prompt will be referred to as the 'Tip Setting Prompt'.

- Third prompt (a way to encourage you to think in steps): Please evaluate if the above code has a CWE-*** vulnerability. Does this code have a CWE-*** vulnerability? (Yes/No). Please rate your confidence in this answer on a scale from 0 to 100, where 100 means very confident and 0 means not confident at all. Let's think step by step.

  This prompt will be referred to as the 'Step-by-Step Prompt'.

3.4. Evaluation Method

To evaluate the results returned by the API, we compared them using a confusion matrix and calculated metrics such as Accuracy, Precision, Recall, and F1 score. The confusion matrix is shown in Table 6, and the formulas for Accuracy, Precision, Recall, and F1 score are as follows:

$$Accurary = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F1\ Score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{4}$$

Table 6. Confusion matrix example for classification evaluation.

		Predicted Class	
		Positive	**Negative**
Actual Class	P (Positive)	TP (True Positive)	FN (False Negative)
	N (Negative)	FP (False Positive)	TN (True Negative)

In Table 6, a True Positive (TP) is when the code has a vulnerability and is detected as vulnerable. A True Negative (TN) is when the code is not vulnerable and is correctly detected as not vulnerable. A False Positive (FP) is when the code does not have a vulnerability but is incorrectly detected as having a vulnerability, and a False Negative (FN) is when code that does have a vulnerability is incorrectly detected as not having a vulnerability. Accuracy is the ratio of correctly predicted instances to the total instances, measuring the overall effectiveness of the model. Precision is the ratio of true positive instances to the sum of true positive and false positive instances, indicating the accuracy of positive predictions. Recall is the ratio of true positive instances to the sum of true positive and false negative instances, measuring the model's ability to identify all relevant instances. The F1 score is the harmonic mean of Precision and Recall, providing a single metric that balances the trade-off between precision and recall.

The process from preparation for the experiment to the derivation of results is illustrated in Figure 2

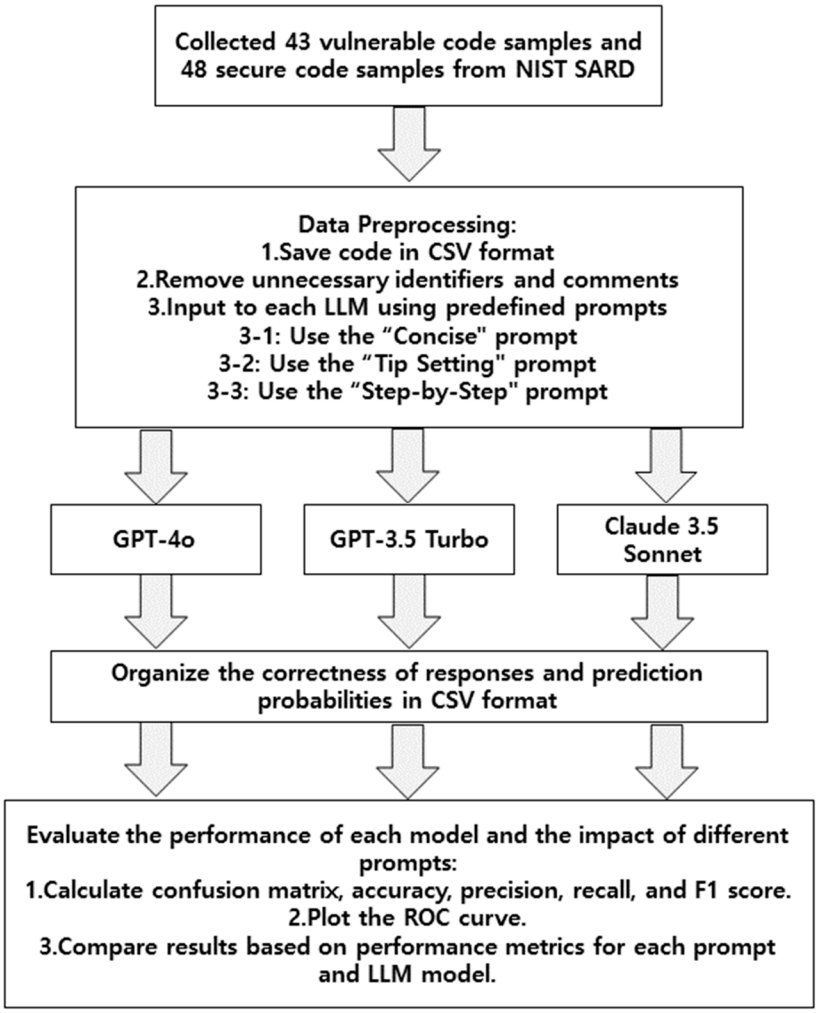

Figure 2. Experimental workflow from preparation to result analysis.

4. Experimental Results

4.1. Analysis of Confusion matrix for GPT-3.5 Turbo

We performed vulnerability analysis on the test cases collected based on GPT-3.5 Turbo's API, and the resulting confusion matrices are shown in Tables 7–9 as prompted. The performance metrics based on these results are shown in Table 10.

Table 7. Results from GPT-3.5 Turbo using the Concise prompt.

		Predicted Class	
		Positive	**Negative**
Actual Class	P (Positive)	42	1
	N (Negative)	40	8

Table 8. Results from GPT-3.5 Turbo using the Tip Setting prompt.

		Predicted Class	
		Positive	**Negative**
Actual Class	P (Positive)	43	0
	N (Negative)	41	7

Table 9. Results from GPT-3.5 Turbo using the Step-by-Step prompt.

		Predicted Class	
		Positive	**Negative**
Actual Class	P (Positive)	42	1
	N (Negative)	44	4

Table 10. GPT-3.5 Turbo's per-prompt performance matrix.

	Accuracy	**Precision**	**Recall**	**F1 Score**
Concise Prompt	0.5495	0.5122	0.9767	0.6720
Tip Setting Prompt	0.5495	0.5119	1.0	0.6772
Step-by-Step Prompt	0.5055	0.4884	0.9767	0.6512

Our experiments found that GPT-3.5 Turbo with Think in Steps prompts produced the least accurate results and the lowest prediction probability. The Set a Tip prompt was not significantly different from the Straight to the Point prompt in terms of the accuracy of results, but it had a higher predictive value. The accuracy of the answers decreased as the number of characters in the entered code increased, with false positives (i.e., it determined that the code was vulnerable when it was not) in all cases where the code was longer than 3000 characters. Overall, the potential for false positives is high, making it a poor tool for vulnerability analysis.

4.2. Analysis of Confusion Matrix for GPT-4o

We performed vulnerability analysis on the test cases collected based on GPT-4o's API, and the confusion matrices derived from the results are shown in Tables 11–13 as prompted. The performance metrics based on these results are shown in Table 14.

Table 11. Results from GPT-4o using the Concise prompt.

		Predicted Class	
		Positive	**Negative**
Actual Class	P (Positive)	36	7
	N (Negative)	21	27

Table 12. Results from GPT-4o using the Tip Setting prompt.

		Predicted Class	
		Positive	**Negative**
Actual Class	P (Positive)	39	4
	N (Negative)	19	29

Table 13. Results from GPT-4o using the Step-by-Step prompt.

		Predicted Class	
		Positive	**Negative**
Actual Class	P (Positive)	43	0
	N (Negative)	9	39

Table 14. GPT-4o's per-prompt performance matrix.

	Accuracy	**Precision**	**Recall**	**F1 Score**
Concise Prompt	0.6923	0.6316	0.8372	0.7200
Tip Setting Prompt	0.7473	0.6724	0.9070	0.7723
Step-by-Step Prompt	0.9022	0.8302	1.0	0.9072

The performance gains from changing the prompts were more pronounced on GPT-4o than on GPT-3.5 Turbo. In GPT-4o, when asked if a particular vulnerability exists for a test case; the system does not simply answer whether the code entered contains the vulnerability, but analyzes the potential vulnerability triggers in the code to answer the question. An effective way to discourage this behavior and ensure that the user only responds to the question asked is to write prompts that encourage step-by-step thinking, which is effective in reducing false positives because it encourages the user to analyze the specific vulnerability being asked about in detail, rather than potential factors, in the following order: the concept of the vulnerability, the cause of the vulnerability, and the analysis of the currently entered code. In the case of the Tip Setting prompts, there was no significant difference in the content of the generated answers when compared to the Straight to the Point prompts. However, we could attempt to improve the accuracy by adding a checking process for the results obtained.

4.3. Analysis of Confusion Matrix for Claude-3.5 Sonnet

We performed vulnerability analysis on the test cases collected based on Claude-3.5 Sonnet's API, and the confusion matrices derived from the results are shown in Tables 15–17 as prompted. The performance metrics based on these results are shown in Table 18.

Table 15. Results from Claude-3.5 Sonnet using the Concise prompt.

		Predicted Class	
		Positive	**Negative**
Actual Class	P (Positive)	40	3
	N (Negative)	14	34

Table 16. Results from Claude-3.5 Sonnet using the Tip Setting prompt.

		Predicted Class	
		Positive	**Negative**
Actual Class	P (Positive)	42	1
	N (Negative)	15	33

Table 17. Results from Claude-3.5 Sonnet using the Step-by-Step prompt.

		Predicted Class	
		Positive	**Negative**
Actual Class	P (Positive)	42	1
	N (Negative)	9	39

Table 18. Claude-3.5 Sonnet's per-prompt performance matrix.

	Accuracy	**Precision**	**Recall**	**F1 Score**
Concise Prompt	0.8132	0.7407	0.9302	0.8247
Tip Setting Prompt	0.8242	0.7368	0.9767	0.8392
Step-by-Step Prompt	0.8901	0.8235	0.9767	0.8933

Performance improvements due to prompt changes were also observed in Claude-3.5 Sonnet. Compared to GPT-4o, the F1 score for the Concise prompt was 0.1047 higher in Claude-3.5 Sonnet. Unlike GPT-4o, Claude-3.5 Sonnet did not show significant performance improvements with the Tip Setting prompt, making it difficult to consider it a meaningful performance enhancement. The reason for this seems to be that while GPT-4o adds a review process when using the Tip Setting prompt, Claude-3.5 Sonnet did not

show significant differences in the content of its responses. For the Step-by-Step prompt, there was a significant performance improvement when compared to the Concise prompt. Like GPT-4o, Claude-3.5 Sonnet tends to perform in-depth analysis of the input code. In this process, it sometimes detects potential vulnerabilities that may not actually exist. By limiting this to step-by-step thinking, it restricts unnecessarily deep analysis, leading to improved performance.

In conclusion, while Claude-3.5 Sonnet's basic performance is superior to GPT-4o, the performance improvements achieved through prompt adjustments are similar to or slightly better in GPT-4o.

4.4. Comparative Performance of the Models

The ROC based on the results of the vulnerability analysis performed by GPT-3.5 Turbo is shown in Figure 3, and the results performed by GPT-4o Turbo are shown in Figure 4. Figure 5 shows the performance of Claude-3.5 Sonnet.

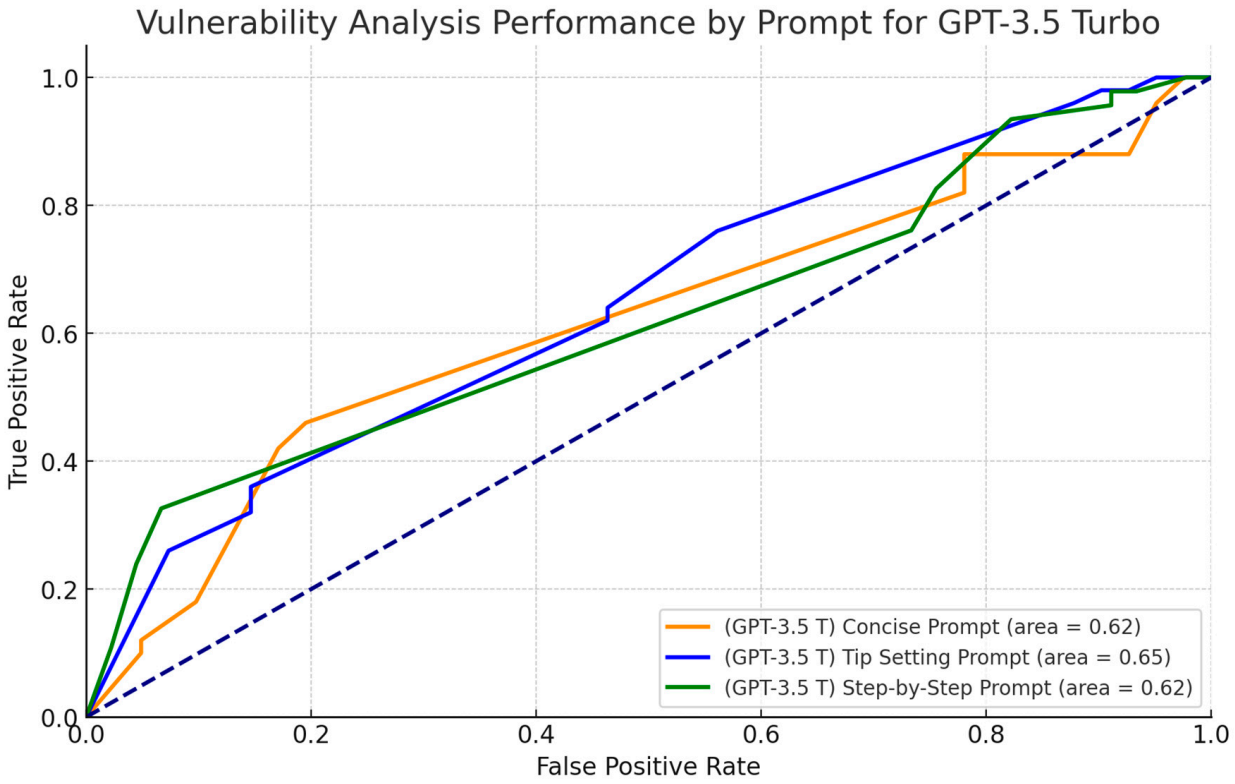

Figure 3. ROC of vulnerability analysis results for GPT-3.5 Turbo.

The area under the curve (AUC) of ROC can be used to evaluate the performance of a classification model. A high AUC value indicates that the model can distinguish between positive and negative classes and can maintain high performance under different threshold settings.

The ROC curves evaluate the sensitivity and false positive rate, while the F1 score evaluates precision and recall. The more extreme the values, the closer the predictive probability is to 0 or 1, and the better the ROC curve will perform.

For GPT-3.5 Turbo, the Concise prompt has an area under the curve (AUC) of 0.62, the Tip Setting prompt has an AUC of 0.65, and the Step-by-Step prompt also has an AUC of 0.62. The predictive probability distribution histogram for GPT-3.5 Turbo, shown in Figure 6, indicates a concentration of predictions around 0.90, with a significant number of predictions also at 1.00, demonstrating a tendency towards high confidence in its predictions.

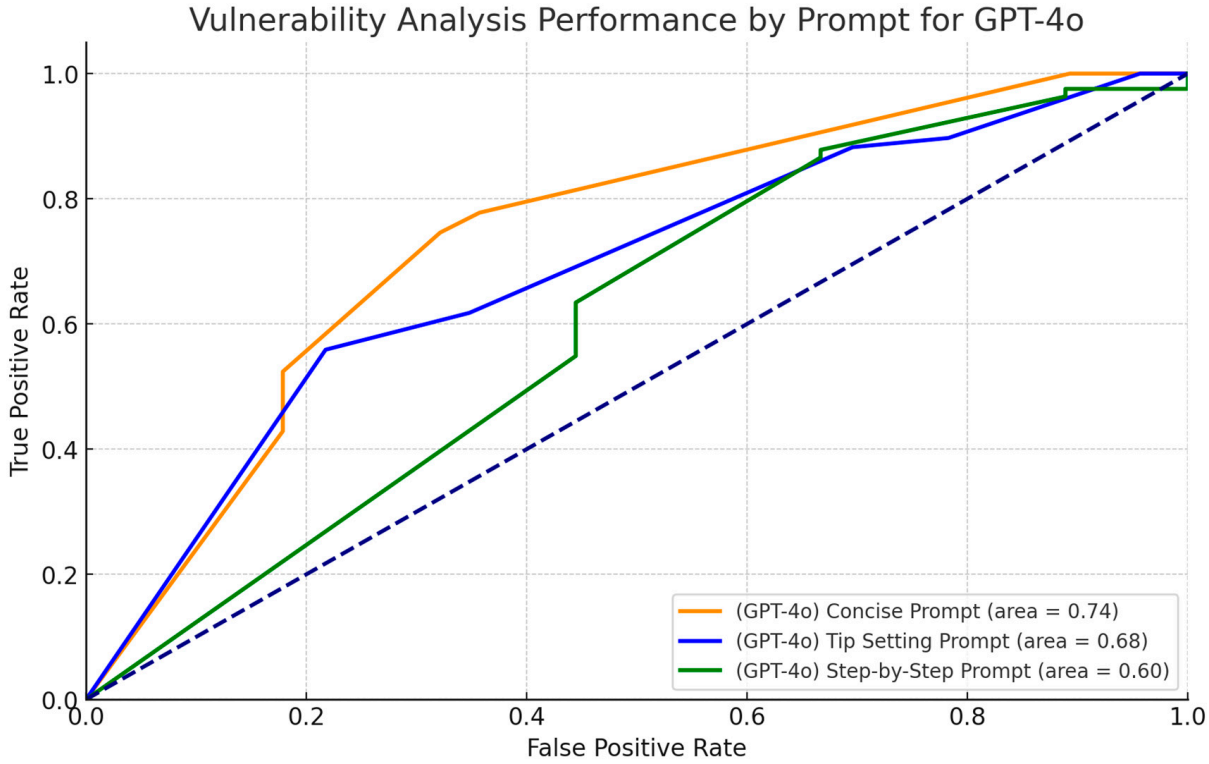

Figure 4. ROC of vulnerability analysis results for GPT-4o.

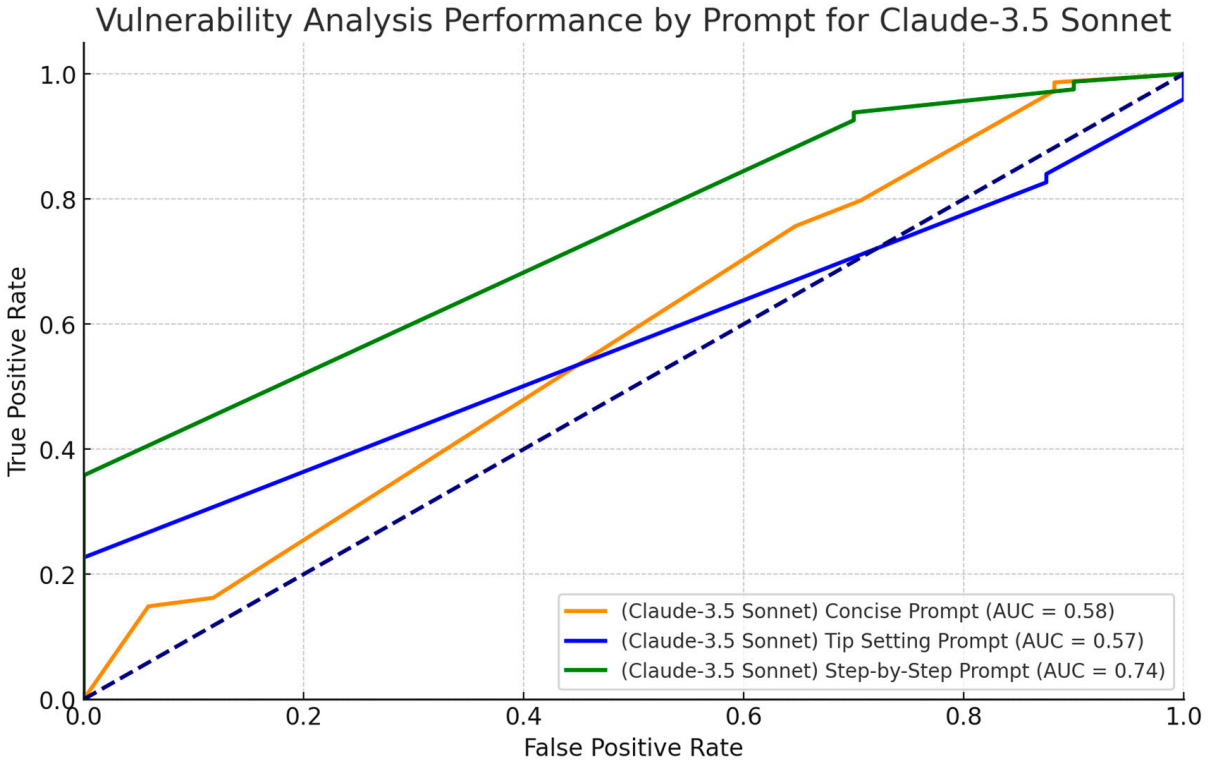

Figure 5. ROC of vulnerability analysis results for Claude-3.5 Sonnet.

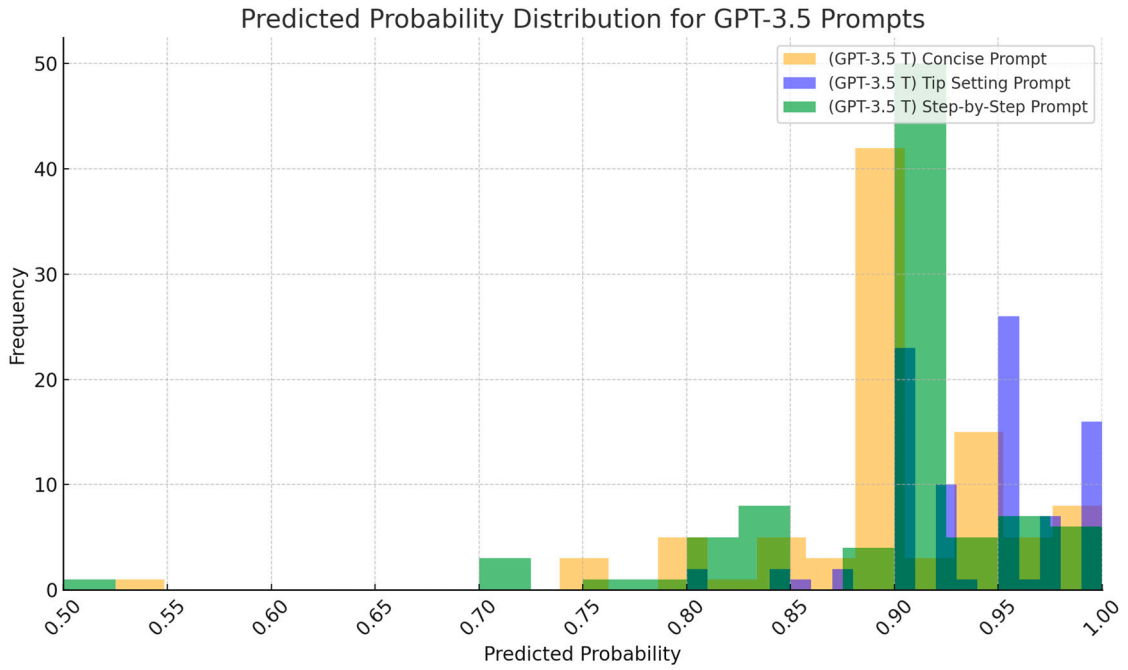

Figure 6. Predicted probability distribution for GPT-3.5 Turbo prompts.

For GPT-4o, the Concise prompt achieves an AUC of 0.74, the Tip Setting prompt has an AUC of 0.68, and the Step-by-Step prompt has an AUC of 0.60. The histogram for GPT-4o, shown in Figure 7, displays a similar pattern, with a high frequency of predictions at the extreme values of 0.95 and 1.00, particularly for the Concise prompt, indicating strong predictive confidence.

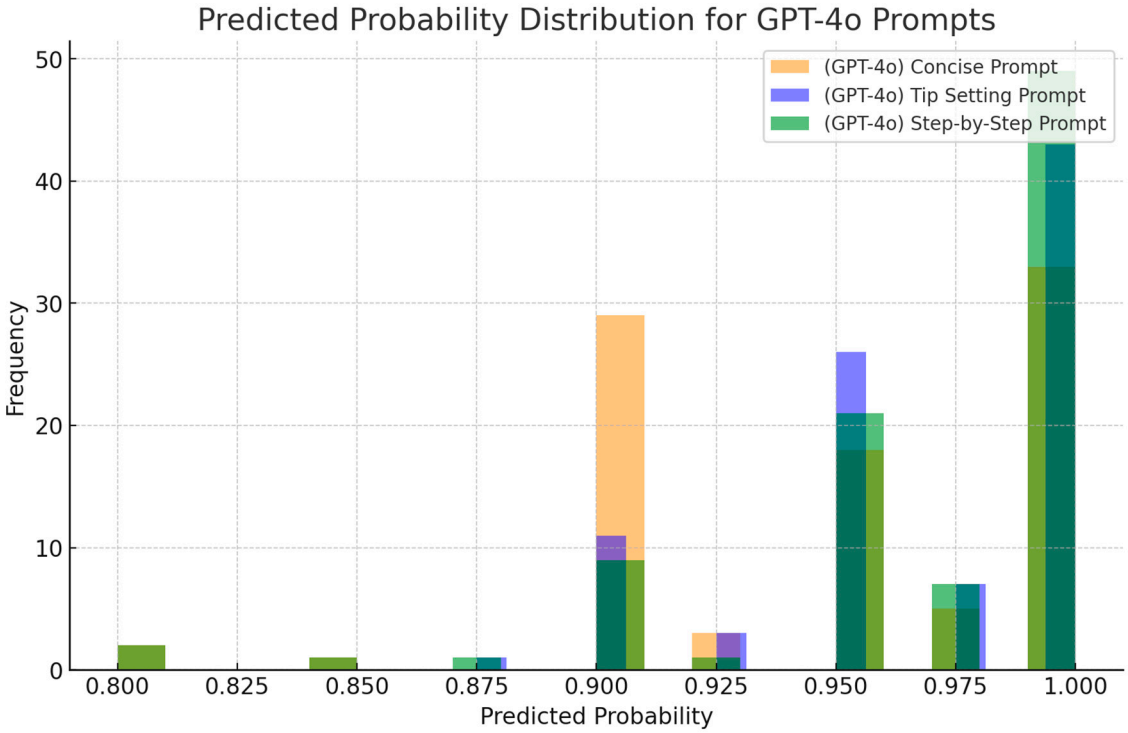

Figure 7. Predicted probability distribution for GPT-4o prompts.

For Claude-3.5 Sonnet, the Concise prompt has an AUC of 0.58, the Tip Setting prompt has an AUC of 0.57, and the Step-by-Step prompt achieves the highest AUC of 0.74.

The histogram for Claude-3.5 Sonnet, presented in Figure 8, shows that the Step-by-Step prompt has a high frequency of predictions at 1.00, while the other prompts have a broader distribution of predicted probabilities. This indicates that the Step-by-Step prompt tends to provide more extreme predictive probabilities, contributing to its higher AUC.

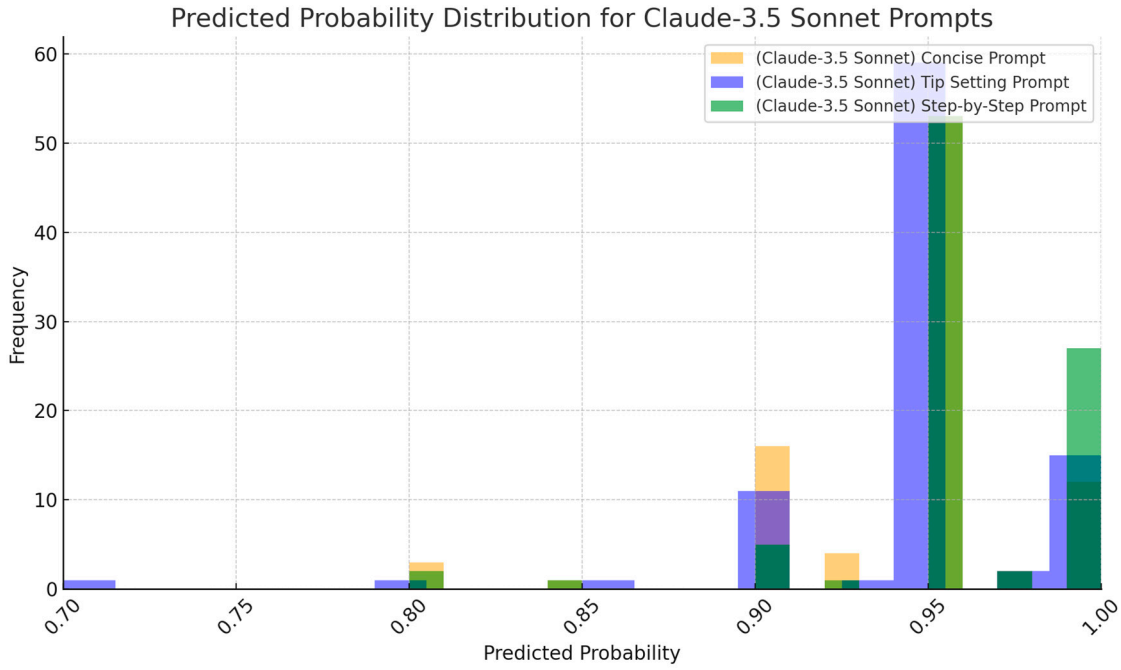

Figure 8. Predicted probability distribution for Claude-3.5 Sonnet prompts.

Based on these results, in situations where finding vulnerable code is crucial, the Step-by-Step prompt applied to Claude-3.5 Sonnet with a high AUC of 0.74 and a high frequency of extreme predictive probabilities is the most effective method. When it is important to detect vulnerabilities in the source code while minimizing false positives and negatives, GPT-4o with the Concise prompt, which also shows high predictive confidence, can be used to achieve balanced performance. Prompts based on GPT-3.5 Turbo can be considered when both the AUC and F1 scores are moderate, but the available budget is limited.

5. Conclusions

The Fourth Industrial Revolution and COVID-19 have increased Internet usage and raised concerns about security in cyberspace. Among them, detecting software vulnerabilities is a fundamental issue in software security, and various techniques have been studied for this purpose. Therefore, this study explores whether GPT-4o and Claude-3.5 Sonnet, which have recently emerged, can be used to analyze software vulnerabilities and compares their performance with GPT-3.5 Turbo. Based on the work of Bsharat, Myrzakhan, and Shen (2023) [16], we also explore whether the performance of vulnerability detection can be improved by simply adjusting the prompts.

The analysis showed that adjusting the simple prompts had a negligible change on GPT-3.5 Turbo, but a significant change on GPT-4o and Claude-3.5 Sonnet. Adding Step-by-Step prompts to GPT-4o significantly improved its performance, increasing the F1 score by 0.1872 from the baseline (Concise prompt), resulting in a high F1 score of 0.9072. For Claude-3.5 Sonnet, the Step-by-Step prompt showed the best performance with an F1 score of 0.8933 and an AUC of 0.74, significantly outperforming its Concise prompt (F1: 0.8247, AUC: 0.58) and Tip Setting prompt (F1: 0.8392, AUC: 0.57). These results suggest that adding Step-by-Step prompts to both GPT-4o and Claude-3.5 Sonnet favors positive class detection, which is useful for finding vulnerabilities in security audits and initial code reviews. However, the combination of GPT-4o and Step-by-Step prompts did not skew

the predictive probability towards extreme values, resulting in a low AUC of 0.60 for ROC. The combination of GPT-4o and Concise prompts had an F1 score of 0.72, lower than the Step-by-Step prompts, but an AUC of 0.74, the highest of all prompts. It is likely useful in situations where false positives and negatives need to be minimized.

Claude-3.5 Sonnet demonstrated a robust performance across different prompt types, with its Step-by-Step prompt achieving the highest overall F1 score among all models and prompts tested. This indicates that Claude-3.5 Sonnet may be particularly well-suited for vulnerability detection tasks, especially when guided by structured prompts.

Finally, the combination of GPT-3.5 Turbo and various prompts did not perform as well as GPT-4o or Claude-3.5 Sonnet, with F1 scores ranging from 0.65 to 0.67 overall and AUCs ranging from 0.62 to 0.65. It also has many false positives for legitimate code, making it difficult to recommend unless resources are limited.

This work is significant because it explores the potential of advanced language models like GPT-4o and Claude-3.5 Sonnet for software vulnerability detection. It also builds on Bsharat, Myrzakhan, and Shen's (2023) [16] work to demonstrate that simple prompt adjustments can effectively change performance to suit specific situations. The superior performance of Claude-3.5 Sonnet, particularly with the Step-by-Step prompt, suggests that it could be a promising tool for code analysis and vulnerability detection.

Future research should focus on maximizing performance through fine-tuning and prompting techniques, as well as exploring the models' capabilities across a wider range of programming languages and vulnerability types. Additionally, investigating the interpretability of these models' decisions could provide valuable insights for improving their reliability in real-world applications. These efforts will help advanced language models become more effective and trustworthy tools for professional code analysis and vulnerability detection.

Author Contributions: Conceptualization, J.B.; Investigation, J.B.; Formal Analysis, J.B. and S.K.; Writing—Original Draft Preparation, J.B.; Writing—Review and Editing, S.M. and S.K.; Translation, S.K.; Supervision, S.M.; Funding Acquisition, S.M.; Project Administration, S.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by The Ministry of Education of the Republic of Korea and the National Research Foundation of Korea (NRF-2022S1A5C2A03093690).

Data Availability Statement: The source code containing the CWEs (Common Weakness Enumerations) used in this study is publicly available at https://samate.nist.gov/SARD/test-cases (accessed on 5 July 2024).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. The Fourth Industrial Revolution. Available online: https://www.sogeti.com/globalassets/global/special/sogeti-things3en.pdf (accessed on 5 June 2024).
2. Lee, M.; Yun, J.J.; Pyka, A.; Won, D.; Kodama, F.; Schiuma, G.; Park, H.; Jeon, J.; Park, K.; Jung, K.; et al. How to Respond to the Fourth Industrial Revolution or the Second Information Technology Revolution? Dynamic New Combinations between Technology, Market, and Society through Open Innovation. *J. Open Innov. Technol. Mark. Complex* **2018**, *4*, 21. [CrossRef]
3. Yang, M.; Zhou, X.; Zeng, J.; Xu, J. Challenges and solutions of information security issues in the age of big data. *China Commun.* **2016**, *13*, 193–202. [CrossRef]
4. Abu Al-Haija, Q. Top-down machine learning-based architecture for cyberattack identification and classification in IoT communication networks. *Front. Big. Data* **2022**, *4*, 78292. [CrossRef] [PubMed]
5. Aslan, Ö.; Aktuğ, S.S.; Ozkan-Okay, M.; Yilmaz, A.A.; Akin, E. A Comprehensive Review of Cyber Security Vulnerabilities, Threats, Attacks, and Solutions. *Electronics* **2023**, *12*, 1333. [CrossRef]
6. Saleous, H.; Ismail, M.; AlDaajeh, S.H.; Madathil, N.; Alrabaee, S.; Choo, K.-K.R.; Al-Qirim, N. COVID-19 pandemic and the cyberthreat landscape: Research challenges and opportunities. *Digit. Commun. Netw.* **2023**, *9*, 211–222. [CrossRef] [PubMed]
7. Jakovljevic, M.; Bjedov, S.; Jaksic, N.; Jakovljevic, I. COVID-19 pandemia and public and global mental health from the perspective of global health security. *Psychiatr. Danub.* **2020**, *32*, 6–14. [CrossRef] [PubMed]
8. Wagner, J.K. Health, housing, and 'direct threats' during a pandemic. *J. Law Biosci.* **2020**, *7*, lsaa022. [CrossRef] [PubMed]

9. Internet Crime Complaint Center (IC3). *2023 Internet Crime Report*; Federal Bureau of Investigation: Washington, DC, USA, 2023. Available online: https://www.ic3.gov/Media/PDF/AnnualReport/2023_IC3Report.pdf (accessed on 27 June 2024).
10. The Evolution of Cyber Threat Intelligence (CTI): 2019 SANS CTI Survey. Available online: https://a51.nl/sites/default/files/pdf/Survey_CTI-2019_IntSights.pdf (accessed on 5 June 2024).
11. Jiang, K.; Li, Y.; Feng, L.; Wang, W. Machine Learning in Cybersecurity: A Review. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2017**, *7*, e1209. [CrossRef]
12. Liu, L.; Xu, B. Research on information security technology based on blockchain. In Proceedings of the 2018 IEEE 3rd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), Chengdu, China, 20–22 April 2018; pp. 380–384. [CrossRef]
13. Prasad, S.G.; Sharmila, V.C.; Badrinarayanan, M.K. Role of Artificial Intelligence based Chat Generative Pre-trained Transformer (ChatGPT) in Cyber Security. In Proceedings of the 2023 2nd International Conference on Applied Artificial Intelligence and Computing (ICAAIC), Salem, India, 4–6 May 2023; pp. 107–114. [CrossRef]
14. Zhou, X.; Zhang, T.; Lo, D. Large Language Model for Vulnerability Detection: Emerging Results and Future Directions. In Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER'24), Pittsburgh, PA, USA, 21–29 May 2024; pp. 47–51. [CrossRef]
15. Chakraborty, S.; Krishna, R.; Ding, Y.; Ray, B. Deep Learning Based Vulnerability Detection: Are We There Yet? *IEEE Trans. Softw. Eng.* **2022**, *48*, 3280–3296. [CrossRef]
16. Bsharat, S.M.; Myrzakhan, A.; Shen, Z. Principled instructions are all you need for questioning LLaMA-1/2, GPT-3.5/4. *arXiv* **2023**, arXiv:16171. Available online: https://arxiv.org/abs/2312.16171 (accessed on 29 May 2024).
17. Chiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F.L. GPT-4 Technical Report. *arXiv* **2023**, arXiv:2303.08774.
18. OpenAI. Available online: https://openai.com/index/hello-gpt-4o/ (accessed on 29 May 2024).
19. Anthropic. Pricing for Claude API. Available online: https://www.anthropic.com/pricing#anthropic-api (accessed on 27 June 2024).
20. Anthropic. Claude 3.5 Sonnet News. Available online: https://www.anthropic.com/news/claude-3-5-sonnet (accessed on 27 June 2024).
21. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Polosukhin, I. Attention Is All You Need. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 62. [CrossRef]
22. Salloum, S.; Gaber, T.; Vadera, S.; Shaalan, K. Phishing Email Detection Using Natural Language Processing Techniques: A Literature Survey. *Procedia Comput. Sci.* **2021**, *189*, 19–28. [CrossRef]
23. LiveBench. Available online: https://livebench.ai/ (accessed on 29 June 2024).
24. Fu, M.; Tantithamthavorn, C.; Nguyen, V.; Le, T. ChatGPT for Vulnerability Detection, Classification, and Repair: How Far Are We? In Proceedings of the 2023 30th Asia-Pacific Software Engineering Conference (APSEC), Seoul, Republic of Korea, 6–9 December 2023; pp. 632–636. [CrossRef]
25. López Espejel, J.; Ettifouri, E.H.; Yahaya Alassan, M.S.; Chouham, E.M.; Dahhane, W. GPT-3.5, GPT-4, or BARD? Evaluating LLMs reasoning ability in zero-shot setting and performance boosting through prompts. *Nat. Lang. Process. J.* **2023**, *5*, 100032. [CrossRef]
26. Ranaldi, L.; Pucci, G. When Large Language Models contradict humans? Large Language Models' Sycophantic Behaviour. *arXiv* **2023**, arXiv:2311.09410.
27. Sosa, D.; Suresh, M.; Potts, C.; Altman, R. Detecting Contradictory COVID-19 Drug Efficacy Claims from Biomedical Literature. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Toronto, Canada, 9–14 July 2023; Association for Computational Linguistics: Baltimore, MD, USA, 2023; pp. 694–713.
28. CWE. Available online: https://cwe.mitre.org/index.html (accessed on 29 May 2024).
29. NIST Software Assurance Reference Dataset (SARD). Available online: https://samate.nist.gov/SARD/ (accessed on 29 May 2024).
30. Seacord, R.C. *Secure Coding in C and C++*, 2nd ed.; Addison-Wesley Professional: Boston, MA, USA, 2013; pp. 1–601.
31. Ministry of Science and ICT; Korea Internet & Security Agency. *Python Secure Coding Guide*, 1st ed.; Ministry of Science and ICT: Sejong, Republic of Korea, 2023; pp. 1–176.