Exploring ChatGPT's Capabilities on Vulnerability Management

Peiyu Liu[1,2,*], Junming Liu[1,2,*], Lirong Fu[3,†], Kangjie Lu[4], Yifan Xia[1,2], Xuhong Zhang[1,5],
Wenzhi Chen[1], Haiqin Weng[6], Shouling Ji[1], and Wenhai Wang[1,2,†]

[1]Zhejiang University, [2]Zhejiang University NGICS Platform, [3]Hangzhou Dianzi University, [4]University of Minnesota,
[5]Jianghuai Advance Technology Center, [6]Ant Group
{liupeiyu, jmliu, fulirong007}@zju.edu.cn, kjlu@umn.edu, {yfxia, zhangxuhong, chenwz}@zju.edu.cn,
haiqin.wenghaiqin@antgroup.com, {sji, zdzzlab}@zju.edu.cn

Abstract

Recently, ChatGPT has attracted great attention from the code analysis domain. Prior works show that ChatGPT has the capabilities of processing foundational code analysis tasks, such as abstract syntax tree generation, which indicates the potential of using ChatGPT to comprehend code syntax and static behaviors. However, it is unclear whether ChatGPT can complete more complicated real-world vulnerability management tasks, such as the prediction of security relevance and patch correctness, which require an all-encompassing understanding of various aspects, including code syntax, program semantics, and related manual comments.

In this paper, we explore ChatGPT's capabilities on 6 tasks involving the complete vulnerability management process with a large-scale dataset containing 70,346 samples. For each task, we compare ChatGPT against SOTA approaches, investigate the impact of different prompts, and explore the difficulties. The results suggest promising potential in leveraging ChatGPT to assist vulnerability management. One notable example is ChatGPT's proficiency in tasks like generating titles for software bug reports. Furthermore, our findings reveal the difficulties encountered by ChatGPT and shed light on promising future directions. For instance, directly providing random demonstration examples in the prompt cannot consistently guarantee good performance in vulnerability management. By contrast, leveraging ChatGPT in a self-heuristic way—extracting expertise from demonstration examples itself and integrating the extracted expertise in the prompt is a promising research direction. Besides, ChatGPT may misunderstand and misuse the information in the prompt. Consequently, effectively guiding ChatGPT to focus on helpful information rather than the irrelevant content is still an open problem.

1 Introduction

Recently, there has been a notable proliferation of powerful large language models (LLMs) owing to the rapid devel-

---

*Peiyu Liu and Junming Liu contributed equally.
†Lirong Fu and Wenhai Wang are co-corresponding authors.

opment of AI techniques [5, 47]. Numerous world-leading companies such as OpenAI, Facebook, Microsoft, and many open-source maintainers have contributed significantly to the development of a large number of LLMs. To date, LLMs have achieved considerable success and have been widely used in diverse domains, including data augmentation [17], code summarization [54], and medical assiatant [36]. Among all the various proposed LLMs, ChatGPT [11], an intelligent human-machine dialogue LLM, has attracted considerable attention. Notably, ChatGPT has achieved a remarkable feat by becoming the fastest-growing app globally, amassing 100 million users within two months of its launch [3].

Existing research works have demonstrated that ChatGPT exhibits outstanding performance in traditional Natural Language Processing (NLP) tasks, including machine translation [28], question-answering [43], and text summarization [51]. Moreover, since programming language, to some extent, shares many analogous characteristics with natural language—both of them are organized through specific grammatical structures and can express certain semantics, researchers turn to utilize ChatGPT for code-related analysis [34, 53]. For instance, Tian et al. explored ChatGPT's capabilities for code generation, program repair, and code summarization [45]. Additionally, Xia et al. proposed to leverage ChatGPT for automated program repair [50]. Moreover, a recent study conducted by Ma et al. investigates ChatGPT's capabilities for understanding program syntax, static behaviors, and dynamic behaviors [34].

Despite the growing body of literature that applies Chat-GPT for general software engineering, its adoption in the security domain remains underexplored. One of the most important areas in the security domain is vulnerability management [16]. Current ChatGPT-focused research merely focuses on several specific tasks within vulnerability management, such as vulnerability fixing [37]. However, vulnerability management encompasses a comprehensive lifecycle that consists of complex phases, each presenting its unique set of challenges [39]. Without a holistic horizon of the whole lifecycle, the capacity of ChatGPT to bolster this vital security process

remains an unknown question.

To fill this research gap, in this paper, we explore: **Can ChatGPT directly assist software maintainers in diverse tasks during the whole vulnerability management process?** We focus on vulnerability management as it is important, presents complexities, and requires significant manual efforts [8]. Given the similarities between code understanding tasks in vulnerability management and NLP, ChatGPT's application in solving these tasks is highly plausible. Concretely, we want to investigate if ChatGPT can achieve capability on par with the state-of-the-art (SOTA) approaches for vulnerability management tasks. Besides, considering the impact of existing prompt engineering methods, we also aim to systematically investigate their effect on ChatGPT's performance. Finally, for the difficulties encountered by ChatGPT for vulnerability management, we aim to shed light on aspects for future exploration. In summary, this paper measures ChatGPT's performance on various vulnerability management tasks from the following three perspectives. **RQ1**: Does ChatGPT achieve capability on par with the SOTAs? **RQ2**: How do prompt engineering methods impact ChatGPT's performance? **RQ3**: What is the promising future direction to improve ChatGPT's performance on each task?

To answer these research questions, we compared ChatGPT's performance to 11 SOTA approaches on 6 vulnerability management tasks, including bug report summarization, security bug report identification, vulnerability severity evaluation, vulnerability repair, patch correctness assessment, and stable patch classification. To achieve this, we first collect the dataset provided by the SOTA approaches of each task. The dataset used in this paper contains 70,346 samples, 19,355,711 tokens in total. By leveraging this dataset, we evaluate ChatGPT's performance for each task with the same metrics used by each SOTA approach. Then, we investigate the influence of different prompt engineering methods. Finally, we analyze ChatGPT's responses to identify the bottlenecks of each task.

Our evaluation and analysis results demonstrate that (1) ChatGPT can outperform the SOTA approaches without being trained specifically for some vulnerability management tasks, especially for tasks related to software document processing, e.g., summarizing bug reports. (2) ChatGPT can achieve comparable capabilities to the SOTA approaches with the help of prompts that contain manual knowledge for some tasks, e.g., security bug report identification. (3) Providing random demonstration examples in the prompt oftentimes achieves limited performance. By contrast, leveraging ChatGPT in a self-heuristic way—extracting expertise from demonstration examples itself and integrating the extracted expertise in the prompt is a promising research direction on some tasks, e.g., vulnerability severity evaluation. (4) Intuitively, the more information provided in the prompt, the better ChatGPT performs. However, our investigation reveals that providing excessive information can lead to misunderstandings and misuse by ChatGPT. Therefore, directing ChatGPT to prioritize

relevant and constructive information over potentially problematic content is a critical area for further research. Our contributions are as follows.

(1) We conduct the first large-scale evaluation of ChatGPT for vulnerability management tasks. The results indicate the desirable prospects of leveraging ChatGPT to assist vulnerability management.

(2) We investigate the influence of various prompt engineering methods for different vulnerability management tasks, which can provide helpful suggestions on designing better prompts to exploit ChatGPT for each task thoroughly.

(3) We uncover the bottlenecks encountered by ChatGPT on vulnerability management and shed light on promising future directions to improve ChatGPT's performance.

2 Background

2.1 Vulnerability Management Process

Vulnerability management constitutes a crucial process in software development, encompassing identifying, classifying, and mitigating vulnerabilities in software products [8, 16]. As shown in Figure 1, a well-established vulnerability management process adopted by prominent software development teams, such as Mozilla [1], involves at least four general phases: issue discovery, vulnerability confirmation, vulnerability fixing, and patch committing.

Issue Discovery. In this phase, an issue *reporter* reports issues through the bug tracking system or the version control system such as Github [2]. Subsequently, a *triager* examines the reported issues for detailed assessment. In real-world scenarios, the *triager* may encounter a substantial influx of bug reports. The thorough evaluation of the entire report is quite time-consuming. Hence, succinct and accurate **bug report summarization** is pivotal for the *triager* to swiftly grasp the essence of the bug [18]. Unfortunately, there usually exist quality concerns of bug report summaries submitted by the *reporters* due to their various professionalism and comprehension of the projects.

Besides, since the vulnerability management process focuses on security issues, efficient **security bug report identification** among an overwhelming number of bug reports is another crucial task for the *triager*. Nevertheless, distinguishing security-related bug reports requires in-depth domain-specific knowledge and considerable human effort, making it a strong demand for automated identification [49, 57].

Vulnerability Confirmation. During the vulnerability confirmation phase, a *triager*, usually a senior developer, is responsible for preliminarily confirming the existence of the reported vulnerabilities. Afterward, the *triager* will assign the bug-fixing task to an appropriate *fixer* in order of severity.

To prevent potential exploitation, all confirmed reports should be assigned timely. However, when confronted with a large number of vulnerabilities, it is impractical to patch
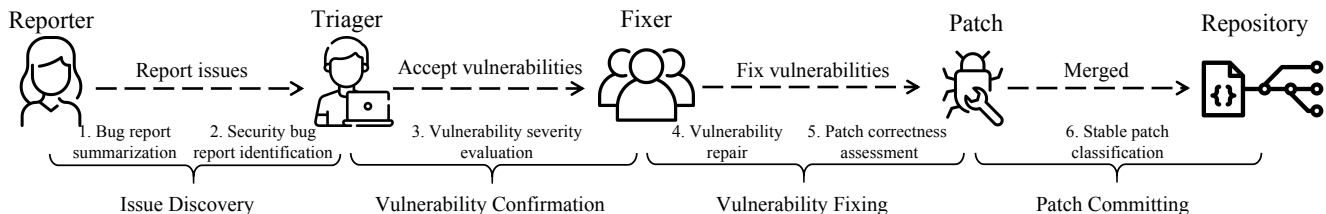
Figure 1: The vulnerability management process.

all of them within a limited time and workforce. Given the limited resources, both the *triager* and the *fixer* need to prioritize more severe vulnerabilities, as they pose higher security risks to the software. Consequently, **vulnerability severity evaluation** becomes the primary step in handling them [48].

Vulnerability Fixing. In this phase, a vulnerability *fixer* generates patches to repair the assigned vulnerabilities. However, patch development demands a profound comprehension of the code context and underlying logic, making it challenging. To support the *fixers* in efficiently fixing software errors, automated **vulnerability repair** tools become crucial as they empower maintainers to produce high-quality patches efficiently [37]. Moreover, the process extends beyond mere repair, considering the prevalence of incorrect patches that fail to address vulnerabilities adequately or introduce new complications [46]. Therefore, **patch correctness assessment** emerges as a critical step in this phase [31].

Patch Committing. Besides the confirmed vulnerability patches generated from the previous process, software maintainers may receive patches from third-party developers, which need to be classified and applied to the suitable codebase for certain users [25]. Typically, software patches can be categorized into stable (bug-fixing) patches and feature enhancement patches. For instance, the Linux kernel maintains a series of stable versions that accept only stable patches. As patches for stable versions contain fixes for bugs that can impact security and stability, it is important to ensure the correctness of **stable patch classification**. However, facing a large number of patches, identifying stable patches accurately and efficiently becomes a significant challenge in this phase.

Summary. Vulnerability management comprises several phases, each presenting its unique set of challenges. Within the scope of these phases, we have pinpointed six pivotal tasks that epitomize the core difficulties encountered in this domain, as listed in Table 1. Meanwhile, developing automatic tools based on traditional software analysis techniques and machine learning models faces significant challenges since these tasks require an in-depth understanding of complex code semantics, program logic, software documents, etc. Considering ChatGPT's established capabilities in code generation and interpretation, it is imperative to investigate its performance across these vulnerability management tasks. Such an investigation could unveil new avenues for employing ChatGPT in this domain and catalyze further research.

Table 1: Tasks, baselines, and dataset. S = Sample. T = Token.

Task	Baseline	Dataset	
		# S	# T
Bug report summarization	iTAPE [18]	33,438	6,176,326
Security bug report identification	Farsec [49] DKG [57] CASMS [35]	22,970	5,686,564
Vulnerability severity evaluation	DiffCVSS [48]	1,642	82,397
Vulnerability repair	LLMset [37] ExtractFix [24]	12	10,601
Patch correctness assessment	Quatrain [46] Invalidator [31] Panther [44]	995 139 208	468,739 31,663 45,204
Stable patch classification	PatchNet [25]	10,896	6,854,217
Total	**11**	**70,346**	**19,355,711**

2.2 ChatGPT and Prompt

ChatGPT is an artificial intelligence chatbot trained to provide human-like responses to users' questions in a conversational way. Users can use ChatGPT through its web interface [11] or official API [15]. A common strategy for applying general models to specific tasks is model fine-tuning. However, this approach is often eschewed due to its labor-intensive nature and significant resource consumption [6]. Consequently, attention has shifted toward optimizing the prompt, i.e., the input of ChatGPT, which significantly influences the relevance and accuracy of the ChatGPT's output [19].

Currently, there are a series of prompt construction strategies employed for enhancing the capability of ChatGPT. Among these, in-context learning has emerged as the dominant paradigm [21]. The foundational approach to in-context learning, known as **0-shot** prompting, instructs ChatGPT by directly describing the task and the associated question [37]. While 0-shot prompting has shown promising performances by leveraging the prior knowledge from the training resource, it sometimes falters when confronted with unfamiliar tasks.

To address this problem, researchers devised advanced prompts by integrating demonstrations, allowing ChatGPT
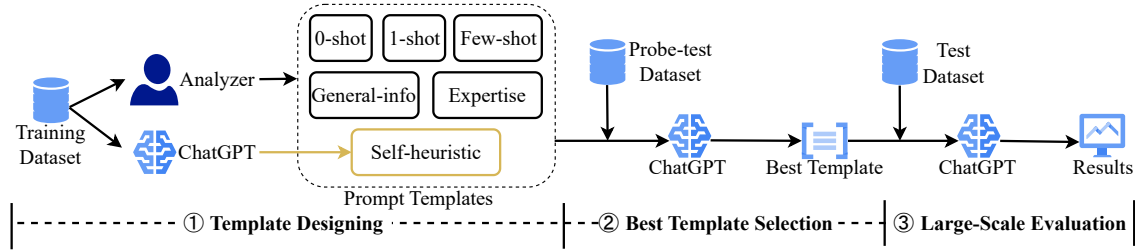
Figure 2: Evaluation pipeline.

to discern patterns inherent to specific tasks [21]. Depending on the volume of demonstration examples within a prompt, they can be categorized into **1-shot** prompting (with a singular demonstration example) or **few-shot** prompting (incorporating multiple demonstration examples) [23]. While well-organized demonstrations have proven effective for simple tasks, they tend to be less efficient for intricate tasks demanding complex logic and domain expertise. To counter these challenges, another line of works enhance ChatGPT by utilizing refined demonstration formatting. This includes the provision of supplemental **general information**, such as role definitions [37, 50], and integrating **domain-specific expertise**, such as vulnerability patterns [42].

In addition to the prompt, the foundational model of ChatGPT also impacts its performance. Users can choose to use gpt-3.5 or gpt-4 when using ChatGPT [15]. A systematic investigation of the effect of different prompts and models for vulnerability management tasks is still lacking. Exploring this problem is essential for thoroughly exploiting ChatGPT for vulnerability management.

3 Evaluation Framework

3.1 Evaluation Pipeline

Figure 2 shows the pipeline of our evaluation, which includes three phases: ① template design, ② best template selection, and ③ large-scale evaluation.

Currently, automatic prompt generation [40] is an ongoing research work that has not been well addressed. Consequently, in phase ①, for each evaluated task, according to the construction rules outlined in §2.2, we first design the prompt templates listed in Table 2 manually based on the heuristics derived from existing widely adopted strategies [7, 23]. To ensure the effectiveness of each prompt template, we assess them using 100 random samples from the training dataset. Subsequently, we refine the templates based on our manual analysis of ChatGPT's responses. Details regarding the template implementation process are discussed in §3.3. As a result, we acquire a collection of prompt templates that demonstrate consistent and reliable performance on the training dataset.

In phase ②, we evaluate all prompt templates with the probe-test dataset to select the best prompt template. To ensure a representative subset of the entire dataset, we construct a probe-test dataset by separating other 10% random samples from the training dataset. However, for certain tasks, the extensive nature of the training dataset used by the SOTA approaches presents challenges. Therefore, for these tasks, considering the time and cost constraints, we limit the probe-test dataset to 1,000 samples. Besides, for vulnerability repairing task, since there is no training dataset used by the SOTA approaches, we opt to construct the probe-test dataset using hand-crafted vulnerabilities provided by [37]. Finally, in phase ③, to fully explore ChatGPT's potential, we use the prompt template that yields the best performance in the probe-test to conduct a large-scale evaluation on the test dataset.[1]

3.2 SOTA Approaches and Dataset

To assess ChatGPT's capabilities on the six vulnerability management tasks described in §2.1, we collect the SOTA approaches for each task as the baselines. The SOTA approaches are derived from the top venues in the fields of security, programming language, and machine learning over the past three years. By sourcing approaches from these reputable venues, we ensure that our evaluation encompasses the most current and relevant advancements in the respective domains, facilitating a comprehensive and up-to-date analysis of ChatGPT's capabilities on vulnerability management. As shown in Table 1 we include 11 SOTA approaches for the six evaluated tasks. Then, we evaluate ChatGPT with the same metrics used by each SOTA approach to conduct the comparison.

To ensure a fair comparison between ChatGPT and SOTA approaches, we obtain the same dataset used in each corresponding SOTA paper for each task. Specifically, we strictly separate the training dataset and test dataset according to the original setting in those papers. Totally, the test dataset used in this paper contains 70,346 samples (19,355,711 tokens).

---

[1]Our evaluations are conducted with ChatGPT official API [15]. Specifically, We use ChatGPT based on gpt-3.5-turbo-0301 in phases ① and ② since it is more economical than gpt-4. The large-scale tests are conducted with ChatGPT based on gpt-4-0314. We set temperature = 0 and top_p = 1.0 to enhance determinism.

Table 2: Templates for task prompt generation.

Template Name	Description	Template
0-shot	Describes the task and directly gives the input query.	**USER** \<task description\> \<input\>
1-shot	Describes the task and provides a random-selected demonstration example before query.	**USER** \<task description\> \<demonstration example\> \<input\>
few-shot	Describes the task and provides multiple random-selected demonstration examples before query.	**USER** \<task description\> \<demonstration example 1\> \<demonstration example 2\> \<demonstration example 3\> \<demonstration example 4\> \<input\>
general-info	Integrates the task-related role assignment into task description through system instruction and complements zero-CoT instructions before query.	**SYSTEM** \<role\> \<task description\> \<reinforce\> **USER** \<task description\> \<task confirmation\> **ASSYSTANT** \<task confirmation\> **USER** \<positive feedback\> \<input\> \<zero-CoT\> \<right\>
expertise	Based on the general-info template, provides manually summarized domain-specific expertise with task description.	**SYSTEM** \<role\> \<task description\> \<expertise\> \<reinforce\> **USER** \<expertise\> \<task description\> \<task confirmation\> **ASSYSTANT** \<task confirmation\> **USER** \<positive feedback\> \<input\> \<zero-CoT\> \<right\>
self-heuristic	Integrates ChatGPT-summarized domain-specific knowledge into task description before query.	**SYSTEM** \<role\> \<task description\> \<reinforce\> **USER** \<knowledge\> \<task description\> \<task confirmation\> **ASSYSTANT** \<task confirmation\> **USER** \<positive feedback\> \<input\> \<zero-CoT\> \<right\>

3.3 Prompt Design and Implementation

In Table 2, we provide detailed explanations on the methodology employed for constructing prompt templates. For 1-shot and few-shot prompts, the demonstration samples are randomly selected from the training dataset. Particularly, according to [23], we select four demonstration samples for few-shot templates to obtain considerably good performance. In the general-information template, we include the instructions that show their superiority in traditional NLP tasks [19], such as zero-CoT [29] and ChatGPT's role [19] in the prompt (Table 12 lists the instructing skills used in the general-info template, which is deferred to Appendix A). As illustrated in Figure 3, the prompt leverages many existing methods, such as the role assignment in line 1. In the expertise prompt template, we provide domain expert knowledge in the prompt. The offered expertise is obtained from the related documentations [4, 12] and literatures [18, 49] (Table 14 provides the expertise content for each task, which is deferred to Appendix A). As shown in Figure 3, the expertise prompt differs from the general-info prompt by providing the characteristic of security bug reports (lines 3 - 6 and 9 - 12). In this example, we guide ChatGPT that memory leak should be treated as a security bug.

It is worth noting that for some evaluated tasks, summarizing expert knowledge can be non-trivial. For example, it is hard to summarize the rules of determining vulnerability severity. In such situations, we turn to explore the potential of guiding ChatGPT by leveraging the knowledge summarized

by itself. Thus, we propose the self-heuristic prompt template. Specifically, we provide several demonstration examples to ChatGPT and ask it to summarize knowledge from these examples. For example, we provide 100 labeled function descriptions to ChatGPT and tell it to *"summarize the characteristics of the function descriptions that should map to the CVSS AV:Network metric"*. Then, we can obtain knowledge summarized by ChatGPT, like "Functions that involve network communication, socket handling, or network device management. Examples: sock_register, udp4_hwcsum, ..." (Figure 6 provides an illustrative example that elucidates the extraction process, which is deferred to Appendix A). After that, we can provide the summary in the self-heuristic prompt in the same way as the expertise prompt. More comprehensive descriptions, including specific examples for each prompt template, are available in Table 13 (deferred to Appendix A). We will provide all the prompts on *https://github.com/Jamrot/ChatGPT-Vulnerability-Management* to support further research.

4 Evaluation Results

In this section, we elaborate on the evaluation results of ChatGPT for the evaluated tasks. For each task, we seek the answers to the research questions proposed in §1 by comparing the performance of ChatGPT and SOTA approaches (**RQ1**), investigating the impact of different prompt templates (**RQ2**), and exploring the potential future research directions to ad-

Table 3: The evaluation result on bug report summarization.

Approach	Prompt	Dataset	ROUGE-1			ROUGE-2			ROUGE-L		
			F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall
iTAPE [18]	-	test	31.36	32.61	31.72	13.12	13.77	13.34	27.79	30.10	29.32
gpt-3.5	0-shot	probe-test	34.33	30.54	42.11	11.05	9.66	13.99	27.95	24.78	34.41
gpt-3.5	1-shot	probe-test	36.82	33.54	43.67	13.27	11.97	16.13	30.86	28.03	35.71
gpt-3.5	few-shot	probe-test	37.30	33.91	44.26	13.99	12.61	16.92	31.52	28.57	37.53
gpt-3.5	general-info	probe-test	32.37	28.23	41.12	10.73	9.25	14.10	26.55	23.10	33.83
gpt-3.5	expertise	probe-test	33.27	29.50	41.23	11.30	9.87	14.37	27.58	24.35	34.32
gpt-3.5	self-heuristic	probe-test	33.08	30.25	40.16	11.26	10.28	13.88	27.53	25.10	33.56
gpt-4	few-shot	probe-test	**40.38**	**39.07**	**44.35**	**15.86**	**15.26**	**17.69**	**34.30**	**33.12**	**37.75**
gpt-4	few-shot	test	39.17	37.52	43.45	14.34	13.58	16.35	33.23	31.77	36.92

```
1   SYSTEM      You are Frederick, an AI expert in bug report analysis. Your
2               task is to decide whether a given bug report is a security bug
3               report (SBR) or non−security bug report (NBR). When
4               analyzing the bug report, take into account that bug reports
5               related to memory leak or null pointer problems should be
6               seen as security bug report. Remember, you're the best AI bug
7               report analyst and will use your expertise to provide the best
8               possible analysis.
9   USER        A security bug report is a bug report describing one or more
10              vulnerabilities of a software. Besides, bug reports that directly
11              mention "memory leak" or "null pointer" problems must be
12              seen as security bug reports. I will  give you a bug report and
13              you will analyze it, step−by−step, to know whether or not it is
14              a security bug report. Got it?
15  ASSISTANT   Yes, I understand. I am Frederick, and I will analyze the bug
16              report.
17  USER        Great! Let's begin then :)
18              For the bug report:
19              <bug report>
20              −−−−−−−−−
21              Is this bug report (A) a security bug report (SBR), or (B) a
22              non−security bug report (NBR).
23              Answer: Let's think step-by-step to reach the right conclusion,
```

Figure 3: An example of the expertise prompt. After removing the **bold pink text**, the rest represents the general-info prompt.

dress the bottlenecks encountered by ChatGPT (**RQ3**).

4.1 Bug Report Summarization

In this evaluation, we ask ChatGPT to summary a given bug report in each query.

ChatGPT's Performance. Table 3 reports the results of this evaluation, which demonstrate that ChatGPT can achieve outstanding performance in this task. For instance, the recall score of ChatGPT based on gpt-3.5 with the 0-shot prompt on the probe-test dataset is 42.11, 13.99, and 34.41 under ROUGE-1, ROUGE-2, and ROUGE-L, respectively, which is better than that of the SOTA approach (31.72, 13.34, 29.32). Moreover, its F1 scores under ROUGE-1 (34.33) and ROUGE-L (27.95) also outperform the SOTA approach (31.36 and 27.79). The results indicate that ChatGPT can gen-

erate high-quantity summaries for bug reports even with the most straightforward prompt (0-shot). This is reasonable since bug report summarization is similar to traditional NLP tasks, such as summarizing news, on which ChatGPT and other LLMs have shown excellent performance [55]. Hence, the results encourage software maintainers to leverage ChatGPT for summarizing bug report and other vulnerability management tasks related to natural language processing.

The Impact of Prompts and Models. From Table 3, we observe that the most straightforward prompt (0-shot) already enables ChatGPT to achieve outstanding performance, whereas more complex prompts do not consistently enhance ChatGPT's performance. Overall, the few-shot prompt templates perform better than other templates in this task. For example, for ChatGPT based on gpt-3.5 evaluated on the probe-test dataset, its ROUGE-L F1 score with the 0-shot prompt is 34.33. This score improves to 37.30 with the few-shot prompt and decreases to 32.37 with the general-info prompt. This observation indicates that providing demonstration examples is helpful for ChatGPT to generate high-quality bug report summarization.

In addition to the prompt evaluation, we also explore the impact of different foundational models. As shown in Table 3, when using the few-shot prompt on the probe-test dataset, the F1 scores of ChatGPT based on gpt-4 are 40.38, 15.86, and 34.30, which are better than that of ChatGPT based on gpt-3.5 (33.08, 11.26, and 27.53). The results encourage us to use gpt-4 when it is available. Moreover, it is important to recognize that reliance solely on automated metrics such as ROUGE does not necessarily guarantee the quality of summarizations from a human reader's perspective. To address this issue, we have also implemented a user study to verify our findings, which is detailed in Section 4.7 .

Implications. In this evaluation, when using the prompt templates that refine demonstration formatting (general-info, expertise, and self-heuristic), most metric scores are even worse than the 0-shot template. This experimental result demonstrates that the original training task of ChatGPT has

Table 4: The evaluation result on security bug report identification. R = Recall. P = Precision. FPR = False Positive Rate. G = G-measure.

Approach	Prompt	Dataset	R	FPR	P	F1	G
DKG [57]	-	test	0.70	0.02	**0.74**	**0.71**	0.81
CASMS [35]	-	test	0.73	0.28	-	-	0.72
Farsec [49]	-	test	0.57	0.16	0.40	0.43	0.64
gpt-3.5	0-shot	probe-test	0.35	0.02	0.21	0.27	0.52
gpt-3.5	1-shot	probe-test	0.76	0.09	0.12	0.21	0.83
gpt-3.5	few-shot	probe-test	0.88	0.06	0.21	0.34	0.91
gpt-3.5	general-info	probe-test	0.29	0.01	0.26	0.28	0.45
gpt-3.5	expertise	probe-test	0.71	0.01	0.57	0.63	0.82
gpt-3.5	self-heuristic	probe-test	0.29	**0.00**	0.56	0.38	0.45
gpt-4	expertise	probe-test	**0.94**	0.04	0.27	0.42	**0.95**
gpt-4	expertise	test	0.68	0.04	0.53	0.57	0.79

already equipped itself with the ability to comprehend this summarization task. Therefore, ChatGPT naturally excels at comprehending, generating, and summarizing text. In this context, providing a straightforward task description and a relevant question can yield strong performance. Conversely, *introducing additional information and expertise in the prompt can lead to confusion for ChatGPT in understanding this task.*

4.2 Security Bug Report Identification

In this evaluation, we ask ChatGPT to answer whether a given bug report is security-related in each query. This evaluation uses three SOTA approaches, DKG [57], CASMS [35], and Farsec [49]. The used metrics are recall, false positive rate (FPR), precision, F1, and G-measure. Among these five performance metrics, recall, precision, f1, and G-measure are the higher, the better, while FPR is the lower, the better.

ChatGPT's Performance. Table 4 shows ChatGPT's performance in this task. Generally, with advanced prompt templates and models, ChatGPT can outperform two baselines, CASMS and Farsec. However, ChatGPT cannot achieve capability on par with DKG. For example, the F1 and G-measure scores of ChatGPT based on gpt-4 with the expertise prompt on the test dataset are 32.6% and 23.4% higher than that of Farsec, respectively. By contrast, these scores are 19.7% and 2.5% lower than that of DKG, respectively. Hence, our investigation indicates that ChatGPT has a certain, yet unsatisfactory, capability to identify security-related bug reports.

The Impact of Prompts and Models. In this evaluation, the impact of prompt templates and models is complicated. First, compared to the 0-shot prompt, some advanced prompt templates increase several metric scores while decreasing others. For example, for ChatGPT based on gpt-3.5 tested on the probe-test dataset, the general-info prompt increases the precision score while decreasing the recall score compared to the 1-shot prompt. We find the reason is that the general-info

prompt makes ChatGPT very "conservative" for this task— ChatGPT only marks 14 tested reports as security-related with only 5 true positives in them. Second, the advanced model, gpt-4, also increases several metric scores while decreasing others. For instance, when using the same expertise prompt on the probe-test dataset, the recall score of ChatGPT based on gpt-4 (0.94) is 32.4% higher than that of ChatGPT based on gpt-3.5 (0.71). However, the precision score decreases by 52.6% at the same time. The results indicate that ChatGPT based on gpt-4 can report more security-related bug reports with a relatively high false positive rate. In practice, this means software maintainers need to spend more time manually checking false positives. With a high recall score and a low precision score, ChatGPT can still benefit the prediction of security bug reports since (1) missing a security-related bug report can lead to severe consequences and (2) it already filters many non-security-related reports.

Implications. When analyzing the results obtained with the 0-shot prompt, we find that ChatGPT exhibits hallucinations regarding the definition of a security bug report. For instance, ChatGPT incorrectly identifies memory leakage (MML) and null pointer dereference (NPD) as not security-related. Providing ChatGPT an MML bug report labeled "security-related" in the 1-shot prompt mitigates this issue and improves the the recall score by 1.17 times. However, the precision score decreases due to other hallucinations, i.e., ChatGPT learns some unrelated information from the demonstration example. For instance, when using the 1-shot prompt shown in Table 13 (deferred to Appendix A), ChatGPT tends to mistakenly mark reports that contain unrelated words of the demonstration example report (e.g., "tab" and "open") as security-related. The results indicate that *when providing demonstration examples, how to make ChatGPT focus on helpful information rather than irrelevant content is an interesting question.*

Furthermore, in the expertise prompt, we directly tell ChatGPT "bug reports related to memory leak or null pointer problems should be seen as security bug reports". Then, for ChatGPT based on gpt-3.5, this prompt achieves the best performance. The results indicate that the summary of domain knowledge can directly benefit the improvement of ChatGPT's performance in security bug report identification.

Failed Cases Analysis. Although the expertise prompt helps mitigate hallucinations, ChatGPT still incorrectly identifies many resource leakage issues as not security-related, causing 17% of the failed cases. This highlights the importance to address this issue in future work. Additionally, in some failed cases, ChatGPT points out that the description in the bug report is not detailed enough, making it difficult to determine whether it is security-related. To further understand what kind of information can help ChatGPT with better identification, we ask ChatGPT what additional information it needs. It indicates that *details about the impact and exploitability of bugs are particularly helpful.*

Table 5: The evaluation result on vulnerability severity evaluation. AV = Attack Vector. AC = Attack Complexity. PR = Privileges Required. UI = User Interaction. R = Recall. P = Precision.

Approach	Prompt	Dataset	AV						AC		PR		UI	
			Network		Adjacent		Physical		High		High		Required	
			R	P	R	P	R	P	R	P	R	P	R	P
DiffCVSS [48]	-	test	0.9242	**0.9384**	0.8750	0.9333	0.8852	0.9153	**0.9151**	**0.9238**	**0.9452**	0.9324	**0.9167**	**0.9296**
gpt-3.5	0-shot	probe-test	0.7143	0.5556	0	N/A	0	N/A	0.4286	0.6923	0.3684	0.5000	0	N/A
gpt-3.5	1-shot	probe-test	**1.0000**	0.2206	0	N/A	0.0909	**1.0000**	0.2857	1.0000	0.1053	**1.0000**	0.2667	0.3077
gpt-3.5	few-shot	probe-test	**1.0000**	0.4285	0.4444	0.6667	0.3636	0.4444	0.6190	0.6842	0.2632	0.3333	0.6667	0.2703
gpt-3.5	general-info	probe-test	0.7857	0.4783	0	N/A	0.1667	0.5000	0.8095	0.3269	0.7368	0.2188	0.4000	0.3000
gpt-3.5	expertise	probe-test	0.8571	0.5714	0.5000	0.6667	0.0833	**1.0000**	0.8095	0.2982	0.5263	0.3704	0.2667	0.2857
gpt-3.5	self-heuristic	probe-test	**1.0000**	*0.7368*	0.7500	**1.0000**	**1.0000**	0.9231	0.8095	0.5484	0.8421	0.6400	0.9333	0.5000
gpt-4	self-heuristic	probe-test	**1.0000**	*0.7368*	**1.0000**	**1.0000**	0.9167	0.9167	0.9048	0.6786	0.8947	0.7083	0.8667	0.7647
gpt-4	self-heuristic	test	0.9848	0.7738	0.9063	0.9355	0.9167	0.8333	0.7961	0.7321	0.8941	0.7917	0.7714	0.8852

4.3 Vulnerability Severity Evaluation

In this evaluation, we ask ChatGPT to map a function to the CVSS exploitability metrics [12] based on its description.

ChatGPT's Performance. As shown in Table 5, Chat-GPT's performance is slightly inferior to the SOTA approach. Specifically, compared to the scores of DiffCVSS, the recall and precision scores of ChatGPT based on gpt-4 with the self-heuristic prompt template are 3.5% and 11.2%, respectively, lower on average. However, ChatGPT can outperform DiffCVSS on several scores. For instance, when mapping AV:Network, the recall of ChatGPT based on gpt-4 with the self-heuristic prompt template (0.9848) is 6.6% higher than that of DiffCVSS (0.9242). Overall, these results show the potential of using ChatGPT in this challenging task.

The Impact of Prompts and Models. Except for the self-heuristic template, ChatGPT's performance with other prompt templates is relatively poor in this task. We take the results on the PR:High metric as an example to demonstrate this investigation. ChatGPT's recall scores on this metric are 61.0%, 88.9%, 72.2%, 22.0%, and 44.3% lower than the SOTA approach (0.9452) when using 0-shot (0.3684), 1-shot (0.1053), few-shot (0.2632), general-info (0.7368), and expertise templates (0.5263), respectively. However, with the self-heuristic prompt templates, ChatGPT's performance significantly improves. For instance, for gpt-3.5, compared to the scores with the expertise template, the recall and precision scores with the self-heuristic template improved by 1.60 and 1.73 times on average, respectively. Besides, the used models also have an essential impact on ChatGPT's performance for this task. With the same self-heuristic prompt, ChatGPT based on gpt-4 outperforms the one based on gpt-3.5 on most scores. Specifically, the precision score of the former is 10.5% higher than the latter on average.

Implications. In fact, ChatGPT struggles with this challenging task. At the beginning of this evaluation, we find that ChatGPT performs poorly with the 0-shot, 1-shot, and few-shot templates. Thus, we attempt to improve its performance in many ways. First, we try to provide more demonstration examples with few-shot prompts. However, due to the maximum token limitation (8,192 tokens for each question and answer), providing many demonstration examples within one prompt is impractical. Inspired by ChatGPT's outstanding NLP capability, we try to compress the demonstration examples with ChatGPT. Consequently, ChatGPT can reduce the length of the demonstration examples by half on average. In this way, we can provide more demonstration examples in one prompt. However, we find that the results remain unsatisfactory.

Furthermore, we notice that providing the description of CVSS metrics in the expertise template slightly improves the performance. Thus, we try to manually adjust the description in the expertise template. Unfortunately, it is hard to construct perfect expertise content for this task. After multiple failures in summarizing the expertise manually, we turn to explore whether ChatGPT can assist us in generating expertise. Specifically, we provide the compressed demonstration examples to ChatGPT and ask it to summarize the characteristics of each CVSS metric. Figure 4 shows the knowledge summarized by ChatGPT. Then, we integrate the summarized knowledge in the self-heuristic prompt. As shown in Table 5, this self-heuristic prompt improves ChatGPT's performance significantly. Hence, we conclude that *extracting expertise by leveraging ChatGPT and incorporating the extracted expertise into prompt is an interesting future research direction, offering the potential to achieve strong performance in challenging vulnerability management tasks.*

Failed Cases Analysis. In this task, we identify two types of failed cases. The first type is due to the inherent ambiguity of the CVSS itself. For example, AV:Network and AV:Adjacent are both related to networks, which makes it challenging for ChatGPT to select the appropriate metric. Similarly, in the AC and PR metrics, overlapping character-

Table 6: The evaluation result on vulnerability repair. Gen = Generated. Vld = compilable. Vuln = Vulnerable. Fn = Functional. Safe = Not Vulnerable. Fixed = Fixed Vulnerabilities. Orig = Using the original code grafting method designed for LLMset [37].

Approach	Prompt	Dataset	# Gen	# Vld	# Vuln	# Fn	# Fn & Vuln	# Fn & Safe	% Vld Repair	# Fixed
ExtractFix [24]	-	test	-	-	-	-	-	-	-	**10**
LLMset [37]	0-shot	test	3,300	674	234	388	252	159	23.6	5
LLMset [37]	expertise	test	3,300	1254	726	926	705	221	17.6	8
gpt-3.5	0-shot	probe-test	350	329	23	166	5	161	48.9	5
gpt-3.5	1-shot	probe-test	350	326	8	176	7	169	51.8	5
gpt-3.5	few-shot	probe-test	350	337	7	145	4	141	41.8	6
gpt-3.5	general-info	probe-test	350	204	4	118	4	114	55.9	4
gpt-3.5 (Orig.)	expertise	probe-test	350	138	40	78	39	39	28.3	5
gpt-3.5	expertise	probe-test	350	259	40	227	39	188	72.6	7
gpt-3.5	self-heuristic	probe-test	350	253	7	153	7	146	57.7	6
gpt-4	expertise	probe-test	350	292	2	290	2	288	**98.6**	7
gpt-4	expertise	test	600	377	20	370	20	350	92.8	**10**

istics related to check conditions can lead to confusion. The second type of failure arises from our rudimentary implementation of the self-heuristic prompt template. For some metrics, our limited sample diversity hinders ChatGPT from forming a well-rounded summary of expert knowledge. For instance, in the self-heuristic samples for the AC:High metric, we overlook the function about checking flags, which leads ChatGPT to fail in summarizing this characteristic and consequently results in incorrect outcomes. Therefore, *when leveraging a self-heuristic prompt template, it is valuable to explore how to extract a variety of samples and guide ChatGPT to make a more comprehensive summary.*

1 *Network: Functions that involve network communication, socket handling, or network device management. Examples: sock_register, udp4_hwcsum, ...*
2 *Adjacent Network: Functions that involve wireless communication, NFC, or Bluetooth. Examples: nfc_start_poll, lib80211_wep_encrypt, ...*
3 *Physical: Functions that involve hardware interaction, device management, or USB handling. Examples: usb_release_dev, snd_card_free, ...*
4 *Not Related: Functions that do not involve any network, adjacent network, or physical interactions, and are related to memory management, page allocation, or other internal system operations. Examples: do_set_mempolicy, do_page_mkwrite, ...*

Figure 4: An example of the knowledge summarized by ChatGPT.

4.4 Vulnerability Repair

In this evaluation, we ask ChatGPT to fix vulnerabilities in the provided code. The baseline is the SOTA approach ExtactFix [24] and six LLMs (named LLMset) evaluated by Pearce et al. [37], including OpenAI's Codex models [5] and AI21's Jurassic-1 models [9]. We use the dataset provided by Pearce et al. [37], which includes 7 hand-crafted vulnerabilities and 12 real-world CVEs. For each vulnerability, the

dataset contains (1) the original buggy file and the vulnerable code snippet, (2) the PoC input that triggered the vulnerability, and (3) the project's regression test. With this dataset, we provide the vulnerable code snippet to ChatGPT and ask it to fix the vulnerability. After ChatGPT generates a response, we graft it with the original buggy file. Then, we use the PoC to test whether the vulnerability has been fixed and the regression tests to ensure that the fix does not break other functionality.

ChatGPT's Performance. LLMset generates 50 responses for each CVE with each LLM (except for AI21, which only generates 300 responses in total due to its comparatively lower API usage limit), resulting in 3,300 generated responses. Similarly, we also generate 50 responses for each CVE with Chat-GPT, resulting in 600 responses with each prompt template. Table 6 summarizes the evaluation results of these responses. First, ChatGPT based on gpt-4 with the expertise prompt can fix 10 / 12 vulnerabilities, which is better than LLMset. Second, we observe that compared to LLMset, ChatGPT can achieve a better valid repair rate (i.e., '# Fn. & Sate' / '# Vld'). Specifically, the valid repair rate of ChatGPT based on gpt-4 with the expertise prompt is 92.8%, which is 4.3 times higher than that of LLMset with the same prompt (17.6%). Additionally, considering that hallucinations may generate syntactically correct patches but do not actually fix the vulnerability, we manually checked the generated patches to filter such potential false positives, and the results indicate the absence of such occurrences.

The Impact of Prompts and Models. In this evaluation, we have the following observations about the impact of prompt templates and models. First, we observe that the 1-shot prompt has a negative effect. Specifically, for ChatGPT based on gpt-3.5, although its valid repair rate with the 1-shot prompt (51.8%) is higher than that with the 0-shot prompt (48.9%), less code generated with the 1-shot prompt is compilable. Besides, the 1-shot prompt does not help ChatGPT fix

Table 7: The evaluation result on vulnerability repair for each CVE. The results are presented as '# Fn & Safe'/'# Vld'. '✓' means that the vulnerability was successfully fixed, while '✗' is the opposite.

Approach	Prompt	EF01 CVE-2016-5321	EF02_01 CVE-2014-8128	EF02_02 CVE-2014-8128	EF07 CVE-2016-10094	EF08 CVE-2017-7601	EF09 CVE-2016-3623	EF10 CVE-2017-7595	EF15 CVE-2016-1838	EF17 CVE-2016-5134	EF18 CVE-2012-5969	EF20 CVE-2018-19664	EF22 CVE-2012-2806
ExtractFix [24]	-	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✗	✓
LLMset [37]	0-shot	33/49	0/2	0/81	-	42/135	4/4	4/65	-	53/58	0/13	0/198	0/69
LLMset [37]	expertise	14/117	23/124	0/205	-	46/78	96/190	11/37	3/98	24/33	0/120	4/171	0/81
gpt-4	expertise	31/38	50/50	-	4/6	0/5	50/50	32/34	2/4	47/50	37/43	47/47	50/50
gpt-4/LLMs/EF	-	✓/✓/✓	✓/✓/✓	✗/✗/✗	✓/✗/✓	✗/✓/✓	✓/✓/✓	✓/✓/✓	✓/✓/✓	✓/✓/✓	✓/✗/✓	✓/✓/✗	✓/✗/✓

more vulnerabilities than the 0-shot prompt (both of them fix 5 vulnerabilities). We also tried to provide more bug patches with the few-shot prompt. Compared to the 1-shot prompt, the few-shot prompt helps gpt-3.5 fix one more vulnerability. However, the few-shot prompt lead to a low valid repair rate. The results indicate that providing existing bug patches to ChatGPT cannot significantly help it generate safe and functional code to fix security vulnerabilities. This observation makes sense because it is hard to fix a vulnerability by following how someone fixes other vulnerabilities since each vulnerable program has its own code semantics.

Second, expert knowledge can help ChatGPT in this hard task. Specifically, the expertise prompt of this task, which provides a comment that describes the vulnerability, e.g., "BUG: stack buffer overflow", can help ChatGPT fix 2 vulnerabilities. Third, gpt-4 has a clear advantage over gpt-3.5 for this task. Specifically, ChatGPT based on gpt-4 generates more compilable code (292) than that based on gpt-3.5 (253). Meanwhile, the valid repair rate of gpt-4 (98.6%) is significantly higher than that of gpt-3.5 (72.6%).

Implications. Table 7 provides more details about the evaluation results for each tested real-world CVE. From Table 7, we observe several interesting findings. First, ChatGPT can fix three CVEs (EF07, EF18 and EF22) which LLMset could not. The fixes of these CVEs are too onerous for LLMset. For instance, EF18's real-world patch is long, removing 10 lines and adding 14; Besides, ChatGPT can fix one CVE (EF20) which ExtractFix could not. ExtractFix failed in this case since it cannot extract this vulnerability's crash-free constraint (CFC). In particular, extracting the CFC with traditional program analysis is quite challenging, which ExtractFix also failed to do in this case. The results indicate that *ChatGPT can be a great choice when traditional program analysis methods fail.*

Second, at the beginning of this evaluation, we find that only a few programs generated based on ChatGPT's response are compilable. The reason is that ChatGPT tends to repeat several code lines before the bug location of the original buggy file in its responses. Thus, the method that grafts LLMset's responses (start at bug locations [37]) with the original buggy

file cannot properly graft ChatGPT's response, making the generated file not compilable due to syntax or semantic errors. For instance, we provide the results of the original code grafting method for ChatGPT based on gpt-3.5 with the expertise prompt in Table 6 (marked with "Orig"). The results reveal that this method cannot generate compilable programs for some tested vulnerabilities. Hence, we developed a new code grafting method that cuts overlap in ChatGPT's response and the original file before the bug location. After that, we obtain the remaining results shown in Table 6, which indicate that more grafted programs are compilable. Although the problem is not caused by the inherent limitation of ChatGPT, we learn an interesting finding from it, i.e., *when using ChatGPT in complex tasks, directly finishing the task with ChatGPT's response can be impractical. Developing workflows to process and leverage ChatGPT's responses appropriately is an important research direction.*

Failed Cases Analysis. In this task, ChatGPT fails to repair two vulnerabilities. For vulnerability EF02_02, ChatGPT's repair attempt differs from the developer's method, which causes the bug location method based on the developer's repair position [37] to be ineffective, making all generated files not compilable. For EF08, the failure is due to insufficient vulnerability-related context provided. In particular, EF08 involves a shift range error, but the context does not include information about the shift variable, leading ChatGPT to "guess" the variable name and thereby failing to generate the correct repair code. Therefore, to further improve ChatGPT's vulnerability repairing capability in real-world applications, *it could be effective to apply more advanced program slicing methods to provide specific vulnerability-related context.*

4.5 Patch Correctness Assessment

In this evaluation, we ask ChatGPT to determine whether a patch correctly fixes a bug. Three SOTA approaches are used as the baselines in this evaluation. We compare ChatGPT with these baselines separately since they use different datasets. Noting that the dataset of Quatrain [46] only contains the description of patches while the datasets of Invalidator [31]

Table 8: The evaluation result on patch correctness assessment (compared with Quatrain [46]).

Approach	Prompt	Dataset	Accuracy	+Recall	-Recall	Precision	F1	AUC
Quatrain [46]	-	test	0.775	0.786	0.773	0.371	0.504	**0.858**
gpt-3.5	0-shot	probe-test	0.617	0.577	0.625	0.246	0.345	0.601
gpt-3.5	1-shot	probe-test	0.682	0.479	0.725	0.270	0.345	0.602
gpt-3.5	few-shot	probe-test	0.720	0.493	0.768	0.311	0.381	0.631
gpt-3.5	general-info	probe-test	0.797	0.359	0.889	0.408	0.382	0.624
gpt-3.5	expertise	probe-test	0.761	0.479	0.821	0.362	0.412	0.650
gpt-3.5	self-heuristic	probe-test	**0.837**	0.366	**0.937**	**0.553**	0.441	0.652
gpt-4	self-heuristic	probe-test	0.789	0.275	0.898	0.364	0.313	0.587
gpt-3.5	desc-code	probe-test	0.725	0.697	0.731	0.355	0.470	0.714
gpt-3.5	code-only	probe-test	0.564	0.817	0.510	0.261	0.396	0.663
gpt-4	desc-code	probe-test	0.700	**0.915**	0.655	0.360	0.517	0.785
gpt-4	code-only	probe-test	0.816	0.901	0.798	0.487	**0.632**	0.850
gpt-4	code-only	test	0.819	0.868	0.811	0.439	0.583	0.840

Table 9: The evaluation result on patch correctness assessment (compared with Invalidator [31]).

Approach	Prompt	Dataset	Accuracy	+Recall	-Recall	Precision	F1	AUC
Invalidator [31]	-	test	0.813	0.900	0.789	0.540	0.675	0.844
gpt-3.5	0-shot	probe-test	0.568	0.758	0.415	0.510	0.610	0.586
gpt-3.5	1-shot	probe-test	0.581	**0.970**	0.268	0.516	0.674	0.619
gpt-3.5	few-shot	probe-test	0.595	0.576	0.610	0.543	0.559	0.593
gpt-3.5	general-info	probe-test	0.608	0.576	0.634	0.559	0.567	0.605
gpt-3.5	expertise	probe-test	0.621	0.545	0.683	0.581	0.563	0.614
gpt-3.5	self-heuristic	probe-test	0.730	0.758	0.707	0.676	0.714	0.732
gpt-4	self-heuristic	probe-test	0.757	0.667	**0.829**	**0.759**	0.710	0.748
gpt-4	self-heuristic	test	**0.849**	0.933	0.826	0.596	**0.727**	**0.880**

and Panther [44] only contain the code of patches.

ChatGPT's Performance. The comparison results with three SOTA approaches are shown in Table 8, Table 9, and Table 10. Generally speaking, with advanced prompts and models, ChatGPT performs comparably to the SOTAs. Specifically, ChatGPT based on gpt-4 outperforms Invalidator on all metric scores. It also outperforms Quatrain and Panther regarding the F1 score. For instance, on the test dataset, the F1 scores of ChatGPT based on gpt-4 are 15.7% and 6.7% higher than Quatrain and Panther, respectively. Moreover, considering the -recall metric that reflects the capability of identifying incorrect patches, gpt-4 exceeds all SOTA approaches on the corresponding test dataset. This indicates that gpt-4 recognizes more incorrect patches, thereby reducing the risk of security instability. All these results highlight the potential of leveraging ChatGPT to assist maintainers in this task.

The Impact of Prompts and Models. From Table 8, we observe that ChatGPT cannot effectively determine patch correctness with straightforward prompt templates. For instance, on the probe-test dataset, the F1 score of ChatGPT based on gpt-3.5 with the 0-shot prompt is 31.5% lower than that of Quatrain. Compared to the 0-shot template, advanced prompt templates increase several metric scores while decreasing

others. However, we notice that all advanced prompt templates can increase the -recall score. Specifically, for gpt-3.5, compared to the 0-shot template (0.625), its -recall score increase 16.0%, 22.9%, 42.2%, 31.4%, and 49.9% when using 1-shot (0.725), few-shot (0.768), general-info (0.889), expertise (0.821), and self-heuristic templates (0.937), respectively. These results indicate that advanced prompt templates can benefit the identification of incorrect patches.

Implications. At the beginning of this evaluation, we find that on the dataset of Quatrain, ChatGPT cannot perform well with all the prompt templates described in Table 2. After analyzing ChatGPT's responses manually, we find that ChatGPT complains that it cannot assess the correctness of patches without code. Hence, we manually collect the corresponding code for each patch and develop a new desc-code template that simultaneously provides the code and description. As shown in Table 8, ChatGPT with the desc-code template still performs somewhat worse than Quatrain.

After manual analysis, we find out an important reason is that *ChatGPT misuses the information in the prompt*. Specifically, when the code and description are provided simultaneously, ChatGPT tends to analyze whether the code changes match the description rather than the correctness of the patch.

Table 10: The evaluation result on patch correctness assessment (compared with Panther [44]).

Approach	Prompt	Dataset	Accuracy	+Recall	-Recall	Precision	F1	AUC
Panther [44]	-	test	0.745	0.811	0.670	0.738	0.773	0.818
gpt-3.5	0-shot	probe-test	0.710	0.963	0.381	0.669	0.789	0.672
gpt-3.5	1-shot	probe-test	0.642	0.972	0.214	0.616	0.754	0.593
gpt-3.5	few-shot	probe-test	0.653	**0.981**	0.226	0.622	0.762	0.603
gpt-3.5	general-info	probe-test	0.720	0.844	0.560	0.713	0.773	0.702
gpt-3.5	expertise	probe-test	0.715	0.771	0.643	0.737	0.753	0.707
gpt-3.5	self-heuristic	probe-test	0.730	0.844	0.583	0.724	0.780	0.714
gpt-4	self-heuristic	probe-test	**0.870**	0.899	**0.833**	**0.875**	**0.887**	**0.866**
gpt-4	self-heuristic	test	0.813	0.829	0.794	0.821	0.825	0.811

For example, ChatGPT tends to report that a patch is incorrect when the patch does not modify a function mentioned in the description. Thus, we further develop the code-only prompt template that removes descriptions from the desc-code template. As shown in Table 8, when using the code-only template, although the performance of ChatGPT based on gpt-3.5 does not increase significantly, ChatGPT based on gpt-4 achieves a comparable performance to Quatrain. The results indicate that *more information is not always better. Guiding ChatGPT to leverage the information in the prompt in a suitable way is an interesting research direction.*

Failed Cases Analysis. When examining false positives, we find that by only analyzing the patch code, ChatGPT often misinterprets the logic error that the patch is supposed to fix. For example, [10] is supposed to fix a problem of incorrect intersection selection in polyhedron sets. However, without the information about the root cause and correct logic, ChatGPT mistakenly assumes that any seemingly reasonable modification corrects the underlying logic error. This issue is also caused by hallucinations, indicating that in the absence of critical information, ChatGPT focuses solely on the surface aspects of the patch and overlooks the essence of the issue it aims to resolve. On the other side, false negatives are mainly caused by unconventional repair patches generated by automatic repair tools. Specifically, rather than directly removing a problematic control block, some automatically generated patches alter the condition within the control block to "if (false)". Therefore, combining the analysis of the implications, we can conclude that more information is not always better, but professional and accurate information is essential. Additionally, while considering removing potential noise in bug reports, it is also necessary to provide more specialized knowledge, such as vulnerability localization and root cause analysis.

4.6 Stable Patch Classification

In this evaluation, we ask ChatGPT whether a given patch (including the description and the code snippet) is a stable patch.

ChatGPT's Performance. Table 11 shows the stable patch classification results, which demonstrate that compared to the SOTA approach, ChatGPT performs slightly worse. Specifically, on the test dataset, for ChatGPT based on gpt-4 with the expertise prompt, its recall score is 4.7% higher than that of PatchNet. However, the remaining scores are 15.0% lower on average. Thus, exploring advanced prompts is still demanded to improve ChatGPT's performance on this task.

The Impact of Prompts and Models. Although ChatGPT cannot achieve capability on par with the SOTA approach for this task, advanced prompt templates can improve its performance. For example, as shown in Table 11, for ChatGPT based on gpt-3.5 tested on the probe-test dataset, its accuracy scores with the expertise (0.762) and self-heuristic (0.646) prompts are 35.6% and 14.1% higher than that with the 0-shot prompt (0.566), respectively.

In this task, gpt-4 and gpt-3.5 each have their advantages. When using the same expertise prompt on the probe-test dataset, ChatGPT based on gpt-4 outperforms ChatGPT based on gpt-3.5 regarding the recall and F1 scores while performing worse on other scores. Considering the application scenario, we suggest that gpt-4 is more suitable for this task for the following reasons. First, recall is more important than precision since identifying as many bug-related patches as possible is critical to ensure the security of the stable version code. Second, gpt-4 achieves a better F1 score, a comprehensive embodiment of precision and recall scores.

Implications. When using the 0-shot, 1-shot, few-shot, and general-info prompts, the precision scores are close to 0.5 while recall scores are close to 1. After manual analysis, we conjecture that the reason is ChatGPT's misunderstanding of what constitutes a stable patch, resulting in a hallucination where any seemingly reasonable patch is assumed stable. By using the expertise prompt with a clear definition of stable patch ("fixing a problem that causes a build error, an oops, a hang, data corruption, a real security issue, or some 'oh, that's not good' issue" [4]), ChatGPT's performance improves significantly. Thus, mitigating ChatGPT's hallucinations is an important research direction to improve its performance on complex tasks.

Table 11: The evaluation result on stable patch classification. ACC = Accuracy. P = Precision. R = Recall.

Approach	Prompt	Dataset	ACC	P	R	F1	AUC
PatchNet [25]	-	test	**0.862**	**0.839**	0.907	**0.871**	**0.860**
gpt-3.5	0-shot	probe-test	0.566	0.564	0.995	0.720	0.508
gpt-3.5	1-shot	probe-test	0.555	0.558	0.986	0.713	0.496
gpt-3.5	few-shot	probe-test	0.557	0.561	0.964	0.709	0.501
gpt-3.5	general-info	probe-test	0.568	0.565	**0.996**	0.721	0.510
gpt-3.5	expertise	probe-test	0.762	0.761	0.837	0.798	0.752
gpt-3.5	self-heuristic	probe-test	0.646	0.631	0.884	0.737	0.614
gpt-4	expertise	probe-test	0.736	0.694	0.945	0.800	0.708
gpt-4	expertise	test	0.733	0.679	0.950	0.792	0.716

Failed Cases Analysis. In actual stable patch classification scenarios, the definition of "stable" is more complex, to some extent exceeding the scope defined in official documentation [4]. For example, some patches that fix minor issues such as "rename helper function" [14] are accepted into the stable branch, whereas some more serious issues like "fix memory leak" [13] are not included. This might be because besides the explicit definition, determining whether a patch is stable also involves considering other deeper factors. Models trained on relevant datasets (i.e., PatchNet [25]) may have acquired deeper knowledge, thus performing better in tasks with complex definitions than ChatGPT. Based on the above observation, *developing methods to extract deep features from existing datasets to augment ChatGPT's vulnerability management capabilities is a promising research direction.*

4.7 User Study

In addition to demonstrating the superior performance of ChatGPT on vulnerability management tasks, it is crucial to assess its effectiveness in real-world scenarios for developers. Accordingly, we have designed a pilot user study that focuses on the bug report summarization process. This study aims to explore the perceptions of various developer groups regarding the bug report summarizations generated by ChatGPT with those produced by the SOTA work, iTAPE [18]. It is important to note that all bug reports used in this study are sourced from open-source bug-tracking systems, ensuring that no confidential information is compromised.

Experiment Settings. We recruit 20 participants and divide them into two groups—experts and intermediates—to explore how the practical value of using ChatGPT for vulnerability management varies with developer experience. The experts group consists of five engineers from the industry, each with more than five years of experience, whereas the intermediates group includes fifteen maintainers from open-source repositories, each with at least three years of development experience. We provide each participant with 100 randomly selected bug report pairs from our test set, each pair consisting of the origi-

nal bug report and its summarization generated by iTAPE and ChatGPT. Participants are instructed to evaluate the quality of each summary based on three criteria: (1) Correctness - The accuracy with which the summarization captures the essence of the bug report. (2) Conciseness - The succinctness of the summarization. (3) Readability - The fluency and clarity of the summarization.

We also provide the instruction "*Please rate on the scale of 1 to 5 (1 - Poor, 2 - Marginal, 3 - Acceptable, 4 - Good, 5 - Excellent)*" to guide the scoring of each report.
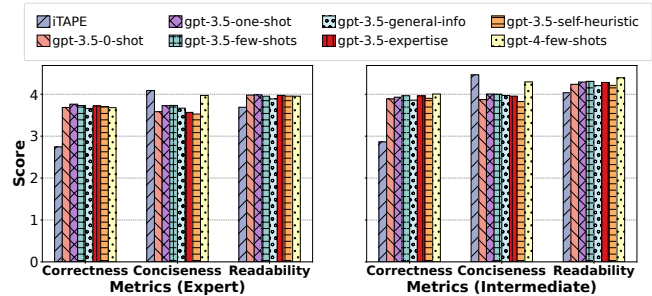

Figure 5: Results of user study (1 - Poor, 2 - Marginal, 3 - Acceptable, 4 - Good, 5 - Excellent).

Results. As depicted in Figure 5, the average scores across different dimensions show that ChatGPT generally produces better summarizations in terms of correctness compared to iTAPE, with highest scores of 3.76 (experts) and 4.01 (intermediates), significantly higher than iTAPE's 2.74 (experts) and 2.86 (intermediates). These results underscore ChatGPT's ability to accurately understand and condense the descriptions of bugs. However, iTAPE's outputs receive higher conciseness scores than those of ChatGPT. This outcome is not unexpected, as iTAPE's outputs often omit essential information needed to fully characterize the bug reports. On average, the bug report summarizations of iTAPE contain 8.5 tokens compared to ChatGPT's 11.3 tokens. Despite providing more detailed information, ChatGPT's summarizations are rated higher in readability than iTAPE's, suggesting that the additional detail enhances comprehension for developers rather than detracting from it. Regarding the scores from different participant groups, the professional group generally awards lower scores than the intermediates group, a trend consistent with findings from previous studies [27]. The LLM's training on extensive datasets allows it to perform like a domain expert, which resonates more with intermediate group participants. However, the professional group's familiarity with the domain means that the incremental information provided by the LLM is less impactful, though they still rate ChatGPT's results well in terms of correctness and readability. Furthermore, no single prompt consistently outperforms others in human evaluations, but gpt-4's outputs tend to be more concise and readable while maintaining high accuracy.

5 Discussion

Threats to Validity. A potential concern regarding Chat-GPT's performance is test sample leakage during training. Detecting such leakage in LLMs is challenging [22]. Existing membership inference approaches for traditional AI models are difficult to apply to LLMs, requiring impractical conditions such as marked data in the training dataset [26] or surrogate model training [52]. Consequently, existing works infer sample existence by examining match rates between ground truth and LLM responses [50], which we also adopted. Specifically, when summarizing bug reports, if test samples had leaked, ChatGPT would generate numerous summaries similar to the ground truth. However, we observe less than 0.1% exact matches. To mitigate the potential limitations of exact matching due to ChatGPT's data processing and response randomization, we also investigate fuzzy matches. By leveraging F1 scores under ROUGE-1, we assess the similarity between ground truth and ChatGPT's responses. The results indicate that less than 3.4% of pairs exhibit a similarity exceeding 70%. In other tasks, good performance with a straightforward 0-shot prompt would be expected if ChatGPT had seen the test samples, yet the results indicate poor performance. We consider these hypotheses relatively evidential.

Another concern is the potential use of test samples to adjust prompts. To mitigate this, we restricted all prompt template modifications to the training data. Specifically, (1) demonstration examples used in the 1-shot, few-shot and self-heuristic prompts come from the training dataset, (2) prompt templates are improved based on mistakes observed in training dataset, and (3) the probe-test dataset is separated from the training dataset. Overall, the test dataset remains untouched during prompt adjustment and probe-testing, ensuring a fair evaluation of ChatGPT's capabilities.

Limitations and Future Works. (1) *Task complexity*. We do not investigate how ChatGPT's performance scales with task complexity or size, which is regarded as future work.

(2) *Prompting techniques*. We manually constructed prompt templates based on prior works in LLM evaluation [34] and our empirical analysis. This manual approach is adopted due to the inherent challenges of automatic prompt engineering (APE), a complex area that holds potential for stimulating research [40]. Moreover, our prompt templates and evaluation pipeline are generally applicable to other LLMs. Addressing adaptability issues is an interesting future research direction.

(3) *Real world interaction scenarios*. Our evaluations are conducted in controlled settings using pre-defined datasets. Exploring scenarios where ChatGPT interacts with actual software development and vulnerability management environments would be insightful. Additionally, integrating ChatGPT across multiple tasks to reflect the interconnected nature of vulnerability management processes would provide a more comprehensive view of its performance.

(4) *Alternative AI approaches*. We focus on evaluating ChatGPT's performance since it is the most popular AI product nowadays. Conducting further evaluations with other LLMs to investigate and compare their performance is considered as future work. Additionally, exploring alternative AI approaches, such as fine-tuning open-sourced models, may address some identified shortcomings, representing another interesting future research direction.

(5) *Hallucination issues*. We identify several hallucinations and conduct measures such as expertise prompts and manual checks to mitigate this problem. However, addressing hallucinations in LLMs remains an open issue for future research.

6 Related Work

AI for Vulnerability Management. With the rapid development of AI, machine learning models have been enthusiastically promoted as tools for various vulnerability management tasks [30, 32, 33, 58]. Some prior works leverage AI methods for automated software artifact generation [56], vulnerability severity examination [48], and bug fixing [38], demonstrating impressive capabilities in these areas. Inspired by this, this paper explores the potential of using ChatGPT, the most emerging AI product currently, for vulnerability management.

The Applications of ChatGPT. ChatGPT has received increasing attention and reputation from various fields. In the beginning, ChatGPT is applied in NLP tasks [20] and has shown impressive capabilities. Considering natural language shares similar aspects as program language, some prior works also explore ChatGPT's capabilities on code analysis [34] and bug fixing [41], indicating the potential for using ChatGPT in software engineering. However, can ChatGPT complete vulnerability management tasks that require a systematic understanding of code syntax, program semantics, and related comments? To the best of our knowledge, it is still an unexplored problem. This paper fills this gap by conducting the first large-scale evaluation of ChatGPT's performance for vulnerability management.

7 Conclusion

This paper conducts the first large-scale evaluation to explore the capabilities of ChatGPT on vulnerability management. Specifically, we compare ChatGPT with 11 SOTA approaches on 6 vulnerability management tasks by using a large-scale dataset containing 19,355,711 tokens. This systematical investigation allows us to understand the capabilities and limitations of ChatGPT on each task. Our findings demonstrate the desirable prospects of leveraging ChatGPT to assist vulnerability management. Meanwhile, we also reveal the difficulties ChatGPT encountered and shed light on future research to explore better ways to leverage ChatGPT in vulnerability management tasks.

Acknowledgments

References

[1] Bugzilla, Oct 2023. https://www.bugzilla.org/.

[2] Github, Oct 2023. https://github.com/.

[3] Has the AI rally gone too far, Oct 2023. https://www.ubs.com/global/en/wealth-management/our-approach/marketnews/article.1592158.html.

[4] Linux. Everything you ever wanted to know about Linux - stable releases, Oct 2023. https://www.kernel.org/doc/html/next/process/stable-kernel-rules.html.

[5] OpenAI Codex, Oct 2023. https://openai.com/blog/openai-codex.

[6] OpenAI. Fine-tuning Document, Oct 2023. https://platform.openai.com/docs/guides/fine-tuning.

[7] OpenAI. Prompt Examples, Oct 2023. https://platform.openai.com/examples.

[8] Security/Firefox/Security Bug Life Cycle, Oct 2023. https://wiki.mozilla.org/Security/Firefox/Security_Bug_Life_Cycle.

[9] AI21. Jurassic-1: Technical Details Evaluation, May 2024. https://www.ai21.com/research/jurassic-1-technical-details-evaluation.

[10] Apache. Fixed intersection selection, May 2024. https://github.com/apache/commons-math/commit/a06a1584.

[11] ChatGPT, May 2024. https://chatgpt.com.

[12] CVSS v3.1 Specification Document, May 2024. https://www.first.org/cvss/v3.1/specification-document.

[13] Linux. Fix memory leak, May 2024. https://github.com/torvalds/linux/commit/6d67726.

[14] Linux. Rename helper functions, May 2024. https://github.com/torvalds/linux/commit/f230d1e8.

[15] OpenAI. API reference, May 2024. https://platform.openai.com/docs/api-reference/chat.

[16] What is vulnerability management, May 2024. https://www.microsoft.com/en-us/security/business/security-101/what-is-vulnerability-management.

[17] Ateret Anaby-Tavor, Boaz Carmeli, Esther Goldbraich, Amir Kantor, George Kour, Segev Shlomov, Naama Tepper, and Naama Zwerdling. Do not have enough data? deep learning to the rescue! In Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, pages 7383–7390, 2020.

[18] Songqiang Chen, Xiaoyuan Xie, Bangguo Yin, Yuanxiang Ji, Lin Chen, and Baowen Xu. Stay professional and efficient: automatically generate titles for your bug reports. In Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering, pages 385–397, 2020.

[19] Benjamin Clavié, Alexandru Ciceu, Frederick Naylor, Guillaume Soulié, and Thomas Brightwell. Large language models in the workplace: A case study on prompt engineering for job type classification. In Natural Language Processing and Information Systems, pages 3–17, Cham, 2023. Springer Nature Switzerland.

[20] Jianyang Deng and Yijia Lin. The benefits and challenges of chatgpt: An overview. Frontiers in Computing and Intelligent Systems, 2(2):81–83, 2022.

[21] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, Lei Li, and Zhifang Sui. A survey on in-context learning. arXiv preprint arXiv:2301.00234, 2023.

[22] Michael Duan, Anshuman Suri, Niloofar Mireshghallah, Sewon Min, Weijia Shi, Luke Zettlemoyer, Yulia Tsvetkov, Yejin Choi, David Evans, and Hannaneh Hajishirzi. Do membership inference attacks work on large language models? arXiv preprint arXiv:2402.07841, 2024.

[23] Shuzheng Gao, Xin-Cheng Wen, Cuiyun Gao, Wenxuan Wang, Hongyu Zhang, and Michael R Lyu. What makes good in-context demonstrations for code intelligence tasks with llms? In 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), pages 761–773. IEEE, 2023.

[24] Xiang Gao, Bo Wang, Gregory J Duck, Ruyi Ji, Yingfei Xiong, and Abhik Roychoudhury. Beyond tests: Program vulnerability repair via crash constraint extraction. ACM Transactions on Software Engineering and Methodology (TOSEM), 30(2):1–27, 2021.

[25] Thong Hoang, Julia Lawall, Yuan Tian, Richard J Oentaryo, and David Lo. Patchnet: Hierarchical deep learning-based stable patch identification for the linux kernel. IEEE Transactions on Software Engineering, 47(11):2471–2486, 2021.

[26] Hongsheng Hu, Zoran Salcic, Gillian Dobbie, Jinjun Chen, Lichao Sun, and Xuyun Zhang. Membership inference via backdooring. In International Joint Conference on Artificial Intelligence (31st: 2022), pages 3832–3838. International Joint Conferences on Artificial Intelligence Organization, 2022.

[27] Peiwei Hu, Ruigang Liang, and Kai Chen. Degpt: Optimizing decompiler output with llm. In Proceedings 2024 Network and Distributed System Security Symposium (2024), volume 267622140, 2024.

[28] Wenxiang Jiao, Wenxuan Wang, JT Huang, Xing Wang, and ZP Tu. Is chatgpt a good translator? yes with gpt-4 as the engine. arXiv preprint arXiv:2301.08745, 2023.

[29] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. Advances in neural information processing systems, 35:22199–22213, 2022.

[30] An Ngoc Lam, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N Nguyen. Combining deep learning with information retrieval to localize buggy files for bug reports (n). In 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pages 476–481. IEEE, 2015.

[31] Thanh Le-Cong, Duc-Minh Luong, Xuan Bach D Le, David Lo, Nhat-Hoa Tran, Bui Quang-Huy, and Quyet-Thang Huynh. Invalidator: Automated patch correctness assessment via semantic and syntactic reasoning. IEEE Transactions on Software Engineering, 2023.

[32] Yi Li, Shaohua Wang, and Tien N Nguyen. Dear: A novel deep learning-based approach for automated program repair. In Proceedings of the 44th International Conference on Software Engineering, pages 511–523, 2022.

[33] Yi Li, Shaohua Wang, Tien N Nguyen, and Son Van Nguyen. Improving bug detection via context-based code representation learning and attention-based neural networks. Proceedings of the ACM on Programming Languages, 3(OOPSLA):1–30, 2019.

[34] Wei Ma, Shangqing Liu, Wenhan Wang, Qiang Hu, Ye Liu, Cen Zhang, Liming Nie, and Yang Liu. The scope of chatgpt in software engineering: A thorough investigation. arXiv preprint arXiv:2305.12138, 2023.

[35] Xiaoxue Ma, Jacky Keung, Zhen Yang, Xiao Yu, Yishu Li, and Hao Zhang. Casms: Combining clustering with attention semantic model for identifying security bug reports. Information and Software Technology, 147:106906, 2022.

[36] Michael Moor, Oishi Banerjee, Zahra Shakeri Hossein Abad, Harlan M Krumholz, Jure Leskovec, Eric J Topol, and Pranav Rajpurkar. Foundation models for generalist medical artificial intelligence. Nature, 616(7956):259–265, 2023.

[37] Hammond Pearce, Benjamin Tan, Baleegh Ahmad, Ramesh Karri, and Brendan Dolan-Gavitt. Examining zero-shot vulnerability repair with large language models. In 2023 IEEE Symposium on Security and Privacy (SP), pages 2339–2356. IEEE, 2023.

[38] Julian Aron Prenner, Hlib Babii, and Romain Robbes. Can openai's codex fix bugs? an evaluation on quixbugs. In Proceedings of the Third International Workshop on Automated Program Repair, pages 69–75, 2022.

[39] Muhammad Shahzad, Muhammad Zubair Shafiq, and Alex X Liu. A large scale exploratory analysis of software vulnerability life cycles. In 2012 34th International Conference on Software Engineering (ICSE), pages 771–781. IEEE, 2012.

[40] Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 4222–4235, 2020.

[41] Dominik Sobania, Martin Briesch, Carol Hanna, and Justyna Petke. An analysis of the automatic bug fixing performance of chatgpt. arXiv preprint arXiv:2301.08653, 2023.

[42] Yuqiang Sun, Daoyuan Wu, Yue Xue, Han Liu, Haijun Wang, Zhengzi Xu, Xiaofei Xie, and Yang Liu. When gpt meets program analysis: Towards intelligent detection of smart contract logic vulnerabilities in gptscan. 2023.

[43] Yiming Tan, Dehai Min, Yu Li, Wenbo Li, Nan Hu, Yongrui Chen, and Guilin Qi. Evaluation of chatgpt as a question answering system for answering complex questions. arXiv preprint arXiv:2303.07992, 2023.

[44] Haoye Tian, Kui Liu, Yinghua Li, Abdoul Kader Kaboré, Anil Koyuncu, Andrew Habib, Li Li, Junhao Wen, Jacques Klein, and Tegawendé F Bissyandé. The best of both worlds: Combining learned embeddings with engineered features for accurate prediction of correct patches. ACM Transactions on Software Engineering and Methodology, 32(4):1–34, 2023.

[45] Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F Bissyandé. Is chatgpt the ultimate programming assistant–how far is it? arXiv preprint arXiv:2304.11938, 2023.

[46] Haoye Tian, Xunzhu Tang, Andrew Habib, Shangwen Wang, Kui Liu, Xin Xia, Jacques Klein, and Tegawendé F Bissyandé. Is this change the answer to that problem? correlating descriptions of bug and code changes for evaluating patch correctness. In Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, pages 1–13, 2022.

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Proceedings of the 31st International Conference on Neural Information Processing Systems, pages 6000–6010, 2017.

[48] Qiushi Wu, Yue Xiao, Xiaojing Liao, and Kangjie Lu. Os-aware vulnerability prioritization via differential severity analysis. In 31st USENIX Security Symposium (USENIX Security 22), pages 395–412, 2022.

[49] Xiaoxue Wu, Wei Zheng, Xin Xia, and David Lo. Data quality matters: A case study on data label correctness for security bug report prediction. IEEE Transactions on Software Engineering, 48(7):2541–2556, 2021.

[50] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. Automated program repair in the era of large pre-trained language models. In Proceedings of the 45th International Conference on Software Engineering (ICSE 2023). Association for Computing Machinery, 2023.

[51] Xianjun Yang, Yan Li, Xinlu Zhang, Haifeng Chen, and Wei Cheng. Exploring the limits of chatgpt for query or aspect-based text summarization. arXiv preprint arXiv:2302.08081, 2023.

[52] Zhou Yang, Zhipeng Zhao, Chenyu Wang, Jieke Shi, Dongsum Kim, Donggyun Han, and David Lo. Gotcha! this model uses my code! evaluating membership leakage risks in code models. arXiv preprint arXiv:2310.01166, 2023.

[53] Tong Ye, Yangkai Du, Tengfei Ma, Lingfei Wu, Xuhong Zhang, Shouling Ji, and Wenhai Wang. Uncovering llm-generated code: A zero-shot synthetic code detector via code rewriting. arXiv preprint arXiv:2405.16133, 2024.

[54] Tong Ye, Lingfei Wu, Tengfei Ma, Xuhong Zhang, Yangkai Du, Peiyu Liu, Shouling Ji, and Wenhai Wang. Cp-bcs: Binary code summarization guided by control flow graph and pseudo code. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, pages 14740–14752, 2023.

[55] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In International Conference on Machine Learning, pages 11328–11339. PMLR, 2020.

[56] Ting Zhang, Ivana Clairine Irsan, Ferdian Thung, Dong-Gyun Han, David Lo, and Lingxiao Jiang. itiger: an automatic issue title generation tool. In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 1637–1641, 2022.

[57] Wei Zheng, JingYuan Cheng, Xiaoxue Wu, Ruiyang Sun, Xiaolong Wang, and Xiaobing Sun. Domain knowledge-based security bug reports prediction. Knowledge-Based Systems, 241:108293, 2022.

[58] Yaqin Zhou, Jing Kai Siow, Chenyu Wang, Shangqing Liu, and Yang Liu. Spi: Automated identification of security patches via commits. ACM Transactions on Software Engineering and Methodology (TOSEM), 31(1):1–27, 2021.

A Appendix

Figure 6: Extracting expertise from demonstration examples.

Self-heuristic Prompt Generation. To improve ChatGPT's performance in intricate tasks demanding domain knowledge,

such as vulnerability severity evaluation, we introduce the self-heuristic prompt. First, we employ ChatGPT to extract expertise from demonstration examples, as shown in Figure 6. Then, we incorporate this expertise into self-heuristic prompts.

Table 12: Prompt skills for the general-info template.

Skill	Description
role	Ask the model to adopt the task-related role.
reinforce	Repeat essential elements of the instructions.
task confirmation	Simulate a task confirmation dialogue.
positive feedback	Provide positive feedback prior to the query.
zero-CoT	Ask the model to think step-by-step.
right	Ask the model to reach the right conclusion.

Skills for the General-info Prompt. Table 12 provides all prompt skills along with their descriptions used for the general-info template. These skills have shown superiority in traditional NLP tasks [19].

Table 14: Task-specific expertise and the referred resources.

Task	Expertise
Bug report summarization	"1. The titles should be within the range of 5 to 15 words, and not contain URLs. 2. At least 30% words of the titles should come from the bug report." [18]
Security bug report identification	"When analyzing the bug report, take into account that bug reports related to memory leak or null pointer problems should be seen as security bug report." [49]
Vulnerability severity evaluation	"CVSS AV metric: 1. Network: The vulnerable component is bound to the network stack and... 2. Adjacent:...attack must be launched from the same shared physical (e.g., Bluetooth)... 3.Physical:..." [12]
Vulnerability repair	"[ERROR_MESSAGE], [VULNERABLE_CODE]." *Note: ERROR_MESSAGE and VULNERABLE_CODE differ by vulnerability.* [37]
Patch correctness assessment	"A correct patch implements changes that "answer" to a problem posed by bug report..." [46]
Stable patch classification	"The patch accepted in Linux-stable release must fix a real and critical bug that causes a build error, an oops, a hang, ..." [4]

Table 13: Templates for task prompt generation.

Name	Template & Example
0-shot	**USER** <task description> <input>
	USER Decide whether a bug report is a security bug report (SBR) or non-security bug report (NBR). Bug report: <bug report> Category:
1-shot	**USER** <task description> <demonstration> <input>
	USER Decide whether a bug report is a security bug report (SBR) or non-security bug report (NBR). Bug report: Memory Leak in about: memory 1. Open a new tab and enter about:memory... Category: SBR ### Bug report: <bug report> Category:
few-shot	**USER** <task description> <demonstration 1> <demonstration 2> <demonstration 3> <demonstration 4> <input>
	USER Decide whether a bug report is a security bug report (SBR) or non-security bug report (NBR). Bug report: Mem leak with IPC::Channel::Channel() in unit_tests UtilityProcessHostTest... Category: NBR ### Bug report: ... Category: NBR ### Bug report: ... Category: NBR ### ... ### Bug report: <bug report> Category:
general -info	**SYSTEM** <role> <task description> <reinforce> **USER** <task description> <task confirmation> **ASSISTANT** <task confirmation> **USER** <positive feedback> <input> <zero-CoT> <right>
	SYSTEM You are Frederick, an AI expert in bug report analysis. Your task is to decide... Remember... **USER** I will give you a bug report and you will... Got it? **ASSISTANT** Yes, I understand. I am Frederick, and I will analyze the bug report. **USER** Great! Let's begin then :) For the bug report: <bug report> Is this bug report (A) a security bug report (SBR), or (B) a non-security bug report (NBR). Answer: Let's think step-by-step to reach the right conclusion,
expertise	**SYSTEM** <role> <task description> <reinforce> **USER** <expertise> <task description> <task confirmation> **ASSISTANT** <task confirmation> **USER** <positive feedback> <input> <zero-CoT> <right>
	SYSTEM You are Frederick... Your task is... memory leak or null pointer... Remember... **USER** A security bug report is... I will give you a bug report... Got it? **ASSISTANT** Yes, I understand. I am Frederick, and I will analyze the bug report. **USER** Great! Let's begin then :) For the bug report: <bug report> ... Let's think step-by-step to reach the right conclusion,
self -heuristic	**SYSTEM** <role> <task description> <reinforce> **USER** <knwoledge> <task description> <task confirmation> **ASSISTANT** <task confirmation> **USER** <positive feedback> <input> <zero-CoT> <right>
	SYSTEM You are Frederick... Your task is... Remember... **USER** Security bug reports (SBR) are... I will ... Got it? **ASSISTANT** Yes, I understand... **USER** Great... For the bug report:... Let's think step-by-step to reach the right conclusion,