Published: 10 March 2026
Accepted: 04 March 2026
Revised: 07 January 2026
Received: 26 December 2024

RESEARCH-ARTICLE

VulRTex: A Reasoning-Guided Approach to Identify Vulnerabilities from Rich-Text Issue Report

ZIYOU JIANG
MINGYANG LI
GUOWEI YANG
LIN SHI
JUNJIE WANG
QING WANG

VulRTex: A Reasoning-Guided Approach to Identify Vulnerabilities from Rich-Text Issue Report

ZIYOU JIANG and MINGYANG LI*, State Key Laboratory of Complex System Modeling and Simulation Technology, China, Science and Technology on Integrated Information System Laboratory, Institute of Software Chinese Academy of Sciences, China, and University of Chinese Academy of Sciences, China
GUOWEI YANG, The University of Queensland, Australia
LIN SHI, School of Software, Beihang University, China
JUNJIE WANG and QING WANG*, State Key Laboratory of Complex System Modeling and Simulation Technology, China, Science and Technology on Integrated Information System Laboratory, Institute of Software Chinese Academy of Sciences, China, and University of Chinese Academy of Sciences, China

Software vulnerabilities exist in open-source software (OSS), and the developers who discover these vulnerabilities may submit issue reports (IRs) to describe their details. Security practitioners need to spend a lot of time manually identifying vulnerability-related IRs from the community, and the time gap may be exploited by attackers to harm the system. Previously, researchers have proposed automatic approaches to facilitate identifying these vulnerability-related IRs, but these works focus on textual descriptions but lack the comprehensive analysis of IR's rich-text information. In this paper, we propose VulRTex, a reasoning-guided approach to identify vulnerability-related IRs with their rich-text information. In particular, VulRTex first utilizes the reasoning ability of the Large Language Model (LLM) to prepare the Vulnerability Reasoning Database with historical IRs. Then, it retrieves the relevant cases from the prepared reasoning database to generate reasoning guidance, which guides LLM to identify vulnerabilities by reasoning analysis on target IRs' rich-text information. To evaluate the performance of VulRTex, we conduct experiments on 973,572 IRs, and the results show that VulRTex achieves the highest performance in identifying the vulnerability-related IRs and predicting CWE-IDs when the dataset is imbalanced, outperforming the best baseline with +11.0% F1, +20.2% AUPRC, and +10.5% Macro-F1, and 2x lower time cost than baseline reasoning approaches. Furthermore, VulRTex has been applied to identify 30 emerging vulnerabilities across 10 representative OSS projects in 2024's GitHub IRs, and 11 of them are successfully assigned CVE-IDs, which illustrates VulRTex's practicability.

CCS Concepts: • **Software and its engineering → Software notations and tools**.

Additional Key Words and Phrases: Large Language Model, Software Vulnerability, Agentic AI

*Corresponding author.

Authors' Contact Information: Ziyou Jiang, ziyou2019@iscas.ac.cn; Mingyang Li, mingyang2017@iscas.ac.cn, State Key Laboratory of Complex System Modeling and Simulation Technology, Beijing, China and Science and Technology on Integrated Information System Laboratory, Institute of Software Chinese Academy of Sciences, Beijing, China and University of Chinese Academy of Sciences, Beijing, China; Guowei Yang, The University of Queensland, Brisbane, Queensland, Australia, guowei.yang@uq.edu.au; Lin Shi, School of Software, Beihang University, Beijing, China, shilin@buaa.edu.cn; Junjie Wang, junjie@iscas.ac.cn; Qing Wang*, wq@iscas.ac.cn, State Key Laboratory of Complex System Modeling and Simulation Technology, Beijing, China and Science and Technology on Integrated Information System Laboratory, Institute of Software Chinese Academy of Sciences, Beijing, China and University of Chinese Academy of Sciences, Beijing, China.

ACM 1557-7392/2026/3-ART
https://doi.org/10.1145/3801149

1 Introduction

Software vulnerabilities widely exist in open-source software (OSS) projects. When developers discover the vulnerabilities, they may submit the IRs to describe the details of these vulnerabilities. To identify these vulnerability-related IRs, security practitioners usually spend a lot of time manually analyzing their contents [39]. They often track these IRs with issue-tracking systems [4, 23], then classify these vulnerability-related IRs with Common Weakness Enumeration (CWE) [6], which categorizes the types of vulnerabilities with their causes, behaviors, and consequences. Some of these identified vulnerabilities will be disclosed in the Common Vulnerabilities and Exposure (CVE) [5], a security database that provides a standardized method to catalog publicly disclosed vulnerabilities, and to alert downstream users in the supply chain to the security risks in the project [1]. However, the manual identification of vulnerability is tedious and time-consuming. The time interval between the creation of vulnerability IRs and the vulnerability disclosure can be exploited by attackers (e.g., zero-day attacks [24]) to harm the system, resulting in millions of dollars of losses in today's businesses [71].

To alleviate this risk at an early stage, researchers have proposed automatic approaches to facilitate the identification of vulnerability-related IRs with their textual description [34, 69, 72]. However, we find that 39.1% of vulnerability-related IRs only have a few-text information in our manual analysis, and they utilize **rich-text information** to describe the vulnerabilities, e.g., page screenshots, video streams, music files, code snippets, etc., which implicitly indicate how the vulnerabilities are triggered, thus helping security practitioners identify vulnerability-related IRs and analyze their CWE-IDs. These previous works focus on textual descriptions but lack a comprehensive analysis of IR's rich-text information, so their practical usage is limited. Among them, we find that over 95% of rich-text IRs utilize page screenshots and code snippets to illustrate how the vulnerabilities are triggered, and their details and contributions are shown as follows:

- **Page Screenshots:** Some developers and users will use page screenshots to display special states during the program's runtime, thereby reflecting possible vulnerabilities in the system. By capturing the specific element in the page screenshots or the transitions between different pages, we may identify which type of vulnerability the project will encounter.
- **Code Snippets:** Some users will provide the simple Proof-of-Concept (PoC) with code snippets to validate the existence of vulnerabilities, and developers will show the vulnerability-related warnings or bug reports with specific code lines in the projects.

To intuitively illustrate how the relationships between these rich-text elements reflect the vulnerabilities, we will introduce them in the following Section 2. From the previous investigations, we focus on exploring how the vulnerabilities are triggered by analyzing the rich-text elements of the page screenshots and code snippets. However, since some information is implicitly described by IR's texts and rich-text elements, it is challenging to identify vulnerability and predict the CWE-IDs:

- **Challenge-1: Difficulty in analyzing the triggering process of vulnerability.** The first challenge comes from how to accurately understand the textual semantics of the rich-text elements and construct the relationships between rich-text elements that describe the triggering process of vulnerabilities. To address it, we utilize the text understanding and reasoning ability of LLMs to construct the relationships between these rich-text elements.
- **Challenge-2: Factual errors in LLM's reasoning process.** The second challenge comes from the factual errors in the reasoning graphs that are inconsistent with the real-world vulnerability triggering process, mainly because the pre-trained data may have some flaws and be outdated. To address it, we incorporate a module to correct the factual errors in the LLM's reasoning steps.
- **Challenge-3: Heavy time cost in LLM's reasoning process.** The third challenge comes from heavy time costs. The security practitioners who manage vulnerability identification need to deal with thousands of IRs. If the efficiency of the automated approach is lower than the manual analysis, its value of practical usage is limited.

For each target IR that needs to identify vulnerabilities, LLM needs to fully explore all the relationships between page screenshots and code snippets to reason how the pages are redirected and determine the vulnerability's type. To improve the effectiveness of reasoning, we utilize the LLM's **Retrieval Augmented Generation (RAG)** in the reasoning process [48], where we can retrieve some relevant cases in the history, then use these cases to guide the reasoning of the target IRs [36]. In practical analysis, we find that vulnerability-related IRs with the same CWE-ID may have commonalities in describing the vulnerabilities. For example, both *Fuel-CMS/issues/536* [2] and *PhpldapAdmin/issues/130* [3] belong to the CWE-79, which utilizes the redirection of page screenshots to show how the attackers set XSS payloads to attack the project. We can treat the first IR as a retrieved case for the second one, which incorporates how the vulnerability is triggered and can guide the vulnerability identification of the second IR.

In this paper, we propose VulRTex, which is a reasoning-guided approach to identify vulnerabilities from IRs with rich-text information. Specifically, it prepares the **Vulnerability Reasoning Database** from historical IRs with LLM, which treats the LLM as an agent [104] that interacts with external tools to understand the semantic information of screenshots and code snippets, and utilize the reasoning ability of LLM [96] to construct the relationships between rich-text elements. The reasoning database contains reasoning graphs that describe how to explore rich-text information in historical IRs to identify whether they contain vulnerabilities. Then, VulRTex incorporates a novel RAG method to retrieve relevant reasoning graphs from the reasoning database that have a similar vulnerability triggering logic to the target IR. Finally, with the retrieved reasoning graphs, VulRTex generates the guidance prompt to guide LLM to identify vulnerabilities.

To evaluate the performance of VulRTex, we conduct experiments on 973,572 IRs with 4,002 vulnerability-related IRs. We compare VulRTex with three types of baselines, and the results show that VulRTex achieves the best performance in identifying the vulnerability-related IRs when the dataset is imbalanced, outperforming baselines with +11.0% F1 and +20.2% AUPRC, with over 2x lower time cost than the baseline reasoning approaches. VulRTex also achieves the best performance in CWE-ID prediction, outperforming the best baseline with +10.5%. Furthermore, VulRTex has been successfully applied on newly tracked IRs across 10 representative projects outside the original dataset after 2024. Among them, 30 emerging vulnerabilities are identified by VulRTex, 11 of them (36.7%) are assigned CVE-IDs, and 9 (30.0%) will potentially be disclosed, which further illustrates its practicality. The major contributions are summarized as follows:

- **Technique**: VulRTex, an automated approach to identify the vulnerability-related IRs. To the best of our knowledge, this is the first work on introducing LLM's reasoning ability to identify rich-text vulnerability-related IRs, as well as using the thought of RAG to reduce the time cost of VulRTex in practical usage.
- **Evaluation**: An experimental evaluation of VulRTex, which shows that VulRTex outperforms all baselines on identifying the vulnerabilities, and the application study on OSS projects further demonstrates its usefulness in practice.
- **Data**: We release the datasets and source code[1] to facilitate the replication and the application of VulRTex in the more extensive contexts.

2 Motivation Example

Recent researchers have conducted a preliminary study on 1,221,677 GitHub IRs in the GHArchive [8], which is a massive IR dataset, archiving the original information of IRs since 2015. Among these IRs, 3,937 of them contain vulnerabilities and have been assigned a CVE-ID, and 3,886 (98.7%) of these vulnerability-related IRs were created earlier than the vulnerability disclosure [72]. Moreover, many developers utilize rich-text information to describe the details of vulnerabilities. To analyze the proportion of these vulnerability-related IRs with rich-text information, we manually inspect these 3,886 IRs created earlier than vulnerability disclosure. We find that 1,520

[1]https://www.github.com/jzySaber1996/VulRTex

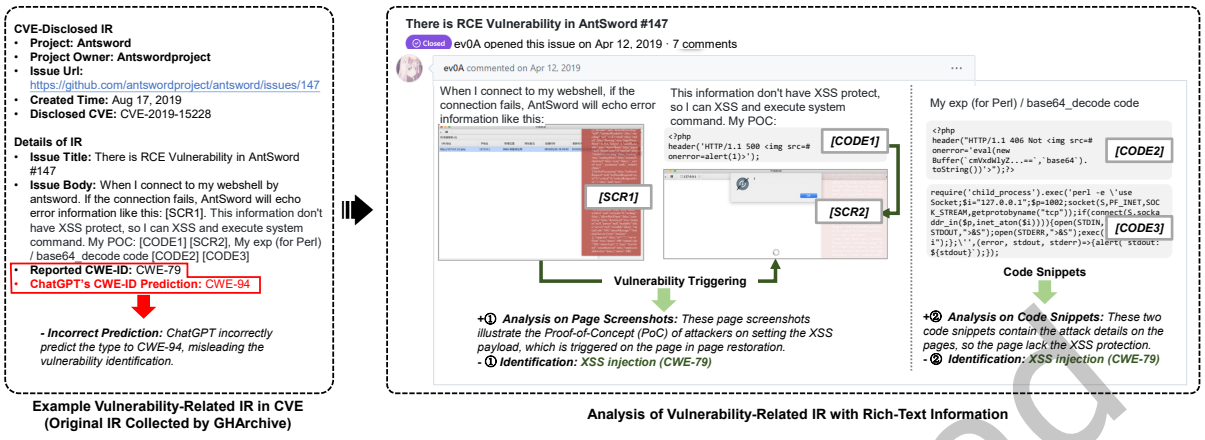


Fig. 1. The vulnerability-related IR with rich-text information, which has been assigned CVE-2019-15228.

(39.1%) contain rich-text elements that relate to the details of vulnerabilities, and over 95% of them are page screenshots and code snippets. Therefore, mining the rich-text information may improve the performance of identifying vulnerability-related IRs, thereby reducing business losses.

Fig. 1 shows an example of vulnerability-related IR that contains rich-text information (*antswordproject/antsword/issues/147*). The author of the project AntSword reports an IR #147 that may have a Remote Code Execution (RCE) vulnerability. This vulnerability relates to multiple vulnerability types in CWE (CWE-79, CWE-94, and CWE-119, etc.), and security practitioners analyze IR's content to determine that this project may encounter the "XSS Injection" (CWE-79) [61]. Different from the other IRs, the author utilizes page screenshots and code snippets to describe this IR, which makes traditional approaches fail to identify it. Besides, we also utilize the widely-used LLM, i.e., ChatGPT [66], to identify the vulnerability from its textual description, and find that ChatGPT incorrectly predicts the type to CWE-94 [62].

To analyze how the rich-text information affects the vulnerability identification, we obtain the original page of this IR. We can see that, the author utilizes two screenshots and three code snippets to describe the triggering process of vulnerability. The process of this attack is as follows: ❶ On the main page of this project ***[SCR1]***, the attacker may set a field as an XSS payload with the PHP script, and attack the project that incorrectly neutralizes user-controllable input. The PoC of this attack is shown in ***[CODE1]***. ❷ If the administrator wants to restore this page, the vulnerable script will be executed, the system command will execute and the XSS payload will be triggered on the page. The page with triggered vulnerability is shown in ***[SCR2]***. To analyze the details of this vulnerability, we can also refer to the code snippets in ***[CODE2]*** and ***[CODE3]***, which illustrate how the attacker utilizes the PHP script to call the system command and perform the XSS injection to the system.

From the above descriptions, we can construct a **reasoning graph** to illustrate how the developers describe the details of this vulnerability. The first path ***[SCR1]→[CODE1]→[SCR2]*** indicates how the vulnerability CWE-79 is triggered, and the second path ***[CODE2]→[CODE3]*** indicates the content of PHP attack script. Based on this reasoning graph, we can determine whether this IR contains vulnerabilities and its relevant CWE-ID. Therefore, we believe that rich-text information can help identify the vulnerability-related IR.

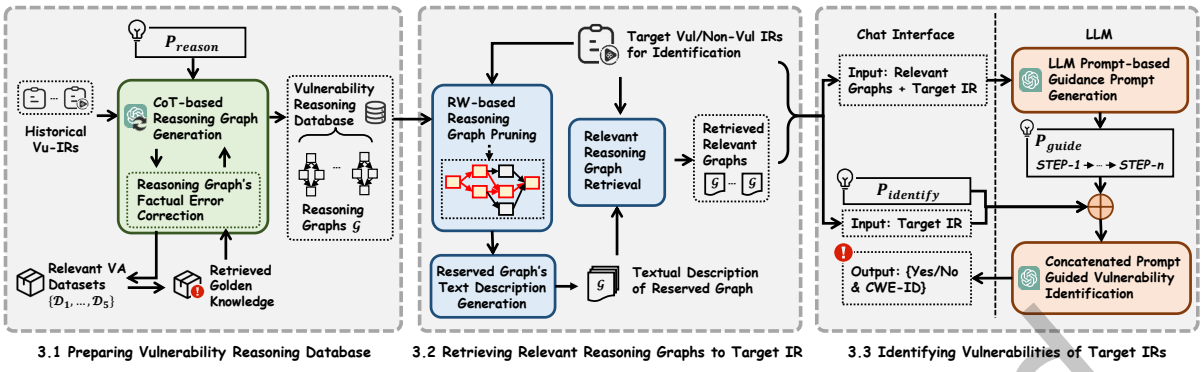


Fig. 2. The overview of our approach VulRTex.

3 Approach

In this section, we introduce the details of VulRTex, and the overall framework is illustrated in Fig. 2. First, VulRTex prepares the vulnerability reasoning database with historical IRs, which contains reasoning graphs $\mathcal{G}$ that describe how the vulnerabilities are triggered based on the rich-text information in historical IRs. Second, to identify the vulnerability from the target IR, VulRTex does not need to reuse the LLMs to explore the IR's rich-text elements. Instead, it retrieves its relevant reasoning graphs from the database to generate prompts to guide LLM to identify vulnerability-related IRs. Afterwards, VulRTex concatenates the guidance prompts, then utilizes this prompt to identify vulnerabilities from target IRs.

3.1 Preparing the Vulnerability Reasoning Database

In Section 2, we discussed that the relationship between rich-text elements that describe the details of vulnerabilities is complex, which can be formulated as a reasoning graph that illustrates steps of exploring rich-text information. Therefore, we prepare the vulnerability reasoning database with reasoning graphs $\mathcal{G}$ from the historical IRs, which treats the LLM as an agent, asking it to utilize tools and reasoning ability to construct relationships between rich-text elements. To implement this logic, we design a specific prompt [95] to control LLM to reason the IR's rich-text information step by step and identify the vulnerabilities.

Moreover, In Huang et al's survey [41], they indicate that the pre-trained data, training, and decoding strategies of LLMs have flaws that result in the inconsistent with real-world facts, which is called LLM's factual error [113]. This factual error is a typical LLM hallucination that exists in the outputs and affects the LLM's practical usage. To bridge this gap, we utilize the following two processes to generate the vulnerability reasoning database:

- **Reasoning Graph Generation:** To realize the LLM reasoning and reduce the time and memory cost, we build a simple **Chain-of-Thought (CoT)** framework, and the generated graphs only contain two types of nodes, i.e., **Observation** and **Action**, and we utilize a specific prompt to guide the historical IR's reasoning graph generation, We also utilize the **inclusion relationship** to analyze texts, screenshots, and code snippets in sequence, which can effectively reduce the nodes and infinite loops in the reasoning graphs.
- **Factual Error Correction:** In each LLM reasoning step, we introduce the external vulnerability awareness (VA) datasets [44, 106] to check and correct the factual errors, which contain the latest vulnerabilities released in the security community and have been perceived and reviewed by experienced security practitioners. This external knowledge is the **golden knowledge** and can be used to reduce the inconsistency in the generated reasoning graph's nodes and edges.

3.1.1 The Definition of CoT-based LLM Reasoning. Given the timestamp t, the CoT-based LLM reasoning incorporates an observation O_t that reflects the result of vulnerability-related IR identification at this timestamp. Then, LLM will conduct an action A_t based on the current observation, which controls the reasoning steps. The policy $\pi(A_t|C_t) = C_t \mapsto A_t$ specifies the way of searching actions, where the C_t is the context sequence of the reasoning steps:

$$C_t = (O_1, A_1, O_2, A_2, ..., O_{t-1}, A_{t-1}, O_t) \tag{1}$$

where policy $\pi(A_t|C_t)$ is determined by the LLM. If the observation O_t indicates that the vulnerability is still unidentified, the LLM will search for an action A_t that can analyze other rich-text information to help the identification; otherwise, the LLM will terminate the reasoning if the vulnerability is identified.

In each action $A_t \in Tools$, it is selected from a set of tools. These tools can effectively parse the texts in screenshots and understand the semantic information of the code snippets. Recently, OpenAI has released tools such as GPT-4o and o1, which can be used to analyze the semantics of multimodal text. However, these models require high time, space, and financial costs, which makes them difficult to use for preparing the reasoning database, and we will not use them in the VulRTex. To conduct the reasoning, we have chosen the text-generative LLMs (i.e., LLaMA, GPT-3, and GPT-3.5), multimodal LLMs (Qwen2VL and GPT-4o), and post-trained LLM (i.e., DeepSeek-R1) as the basic models, and combine them with the toolset. We also incorporate a new inner tool **AgentTerminator** to stop the reasoning when the LLM cannot obtain the new knowledge to continue the reasoning steps. In the finally generated CoTs, the formal description of the observation and actions is as follows:

- **Observation:** `[Vul_Type]|None+[Current_Obs]+[Next_Act]`, where the `[Vul_Type]` is the decision of the CWE-based vulnerability type (e.g., "*Yes, vulnerability is CWE-79.*" in Fig. 3). If the reasoning process is not terminated, the vulnerability type is not determined, so the text is `[None]`. The `[Current_Obs]` and `[Next_Act]` are the texts of the current observation based on the history reasoning steps C_t, and `[Next_Act]` is the action A_t that needs to be conducted in the near future. To make the LLMs understand the CoT, we use plain text to describe the previous two elements.
- **Action:** `[Tool_Name]+([R_Elem]|None)`, where the `[Tool_Name]` is what is called in the agent's tool list, which will be described in the next section. `[R_Elem]` is the tool's parameter that analyzes rich-text information. If the agent has identified the vulnerability type, the reasoning is terminated, so the parameter is `None`.

The reasoning graph $\mathcal{G}$ includes all the observations and actions that identify vulnerability-related IRs. At the timestamp $t \in \{1, ..., T\}$, the paths of $\mathcal{G}$ is formulated as follows:

$$C_{t,i} = (O_1, A_{1,i}, O_{2,i}, ..., O_{t,i}), \qquad \mathcal{G} = \{C_{T,1}, ..., C_{T,m}\} \tag{2}$$

where $C_{t,i}$ is the i_{th} path in the graph. Finally, the graph can be formulated as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{O_1, O_2, ..., O_n\}$ indicates the observations, and $\mathcal{E} = \{A_1, A_2, ..., A_n\}$ indicates the actions that control the steps of reasoning. As is shown in Fig. 3, the reasoning graph $\mathcal{G}$ has four paths, seven observations, and 10 actions to identify the vulnerabilities.

3.1.2 CoT-based Reasoning Graph Generation. The historical IR incorporates the $\{Title, Body\}$, where $Title$ summarizes the main topic of this IR, and $Body$ contains a set of sentences with rich-text information that describes the details of IR. The CoT-based reasoning incorporates a specific prompt to ask the LLM to analyze this content and generate the reasoning graph $\mathcal{G}$.

The prompt controls the graph generation by asking the LLM to think *step by step*. In each reasoning step, we ask the LLM to select **multiple** rich-text elements to analyze whether the IR contains vulnerabilities, as is shown in Fig. 3's observation O_1 (i.e., ***[SCR1]~[SCR4]***, and edges are $\{(O_1 \rightarrow O_{2.1}), ..., (O_1 \rightarrow O_{2.4})\}$). To analyze each of the selected elements, VulRTex will continuously conduct multiple actions based on the observation until it

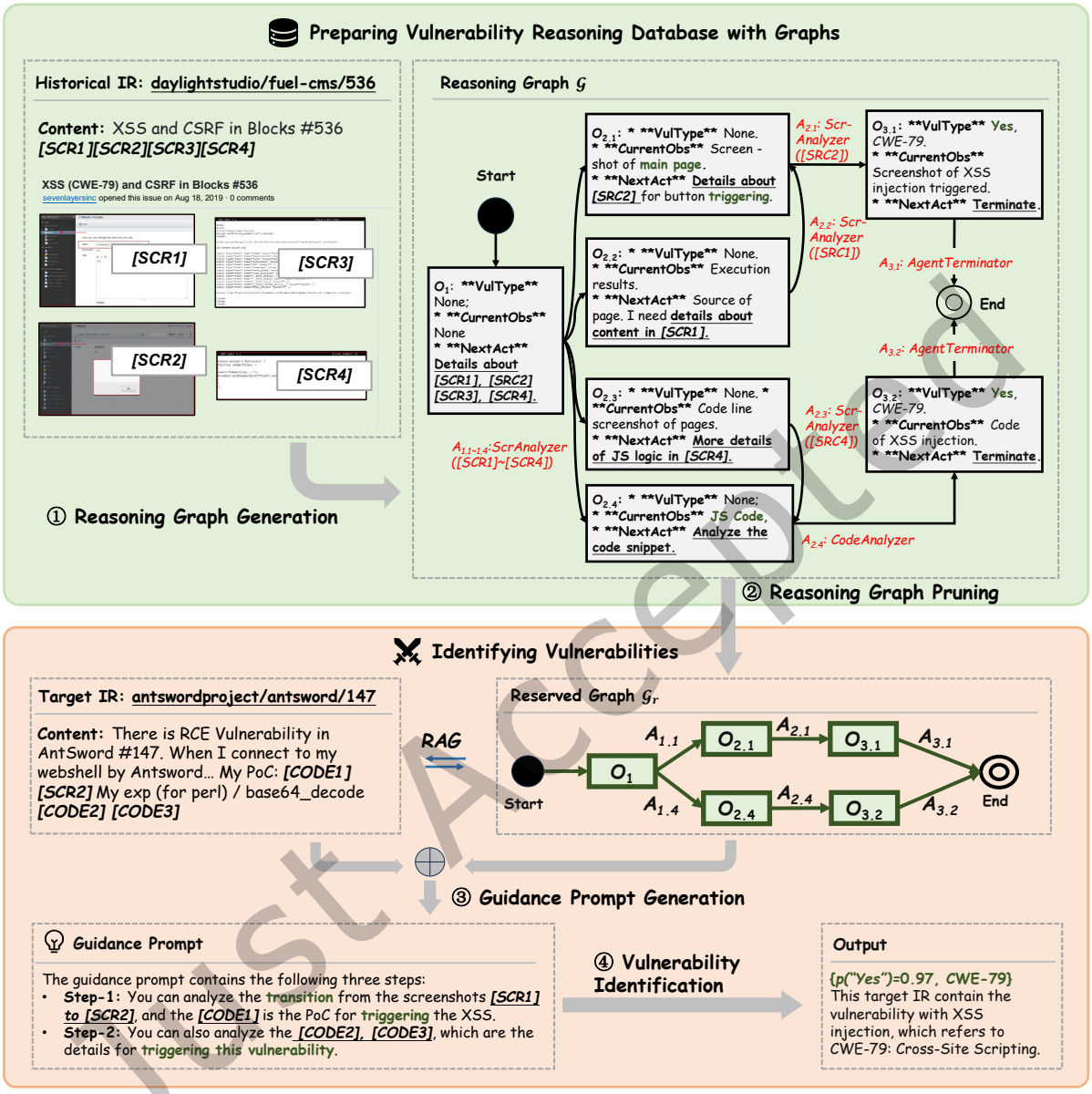


Fig. 3. The steps of identifying target IRs (i.e., Fig. 1's example) vulnerability with rich-text information.

terminates. Therefore, any node representing the observations may be connected to multiple edges representing the actions. The output is a reasoning graph rather than a single-directional path.

In addition, we also define the observation and action in this prompt, as well as the tool list for the action selection. The details of these three tools are shown as follows:

- **ScrAnalyzer:** We utilize Tencent Cloud's OCR [89] to analyze all the page elements in the screenshots ***[SCR]***. Compared with other OCRs, Tencent Cloud can analyze more page elements with high accuracy&efficiency.

- **CodeAnalyzer:** We utilize the CAST [84] to generate the description of code ***[CODE]***, which is a novel method to analyze codes with Abstract Syntax Trees (AST) [108].
- **AgentTerminator**: We ask the LLM to terminate the agent.

The following prompt P_{reason} specifies how to find the vulnerabilities step by step:

P_{reason}: **Prompt for Generating Reasoning Graph**
 # Steps for Generating Reasoning Graph Please think **step by step**. For each step, you need to select **multiple** rich-text elements that relate to the vulnerability. Then, you should identify whether this IR contains the vulnerability and output an "**Observation**" based on context information. It would be best to choose the "**Action**" from "**Tools**" to control the reasoning into the next "Observation" after thinking. - ***Input:*** *The content of historical IR and the rich-text information.* - ***Input:*** *The definition of "Observation" and "Action".* * ***Note of Inclusion Relationship:*** *We suggest you first analyze the text, then explore the page screenshot* ***[SCR]****, and finally analyze the code snippets* ***[CODE]****.*

Moreover, we find that IRs allow the **inclusion relationship** $\textbf{\textit{Text}} \xrightarrow{Inc} \textbf{\textit{[SCR]}} \xrightarrow{Inc} \textbf{\textit{[CODE]}}$ among rich-text information. This relationship comes from our investigation of issues from GitHub, GitLab, and other OSS platforms, which means IR's text contains screenshots and code snippets, and some screenshots contain code. Therefore, we suggest analyzing the comment, screenshot, and code in order based on the greedy search. Our manual analysis found that the reasoning graphs generated by the VulRTex have 3 fewer nodes on average with the inclusion relationship, which means the VulRTex can explore more elements with fewer iterations. We iteratively generate the observations and actions, until the reasoning graph is determined.

3.1.3 Reasoning Graph's Factual Error Correction. As illustrated in the previous section, the LLM factual errors may affect the reliability of reasoning graphs. We select the five representative VA datasets $\mathcal{D}$ in Table 1 to correct the factual errors, To obtain more comprehensive and up-to-date external knowledge, we utilize the following three criteria to select the VA dataset:

- **Criterion-1: Inclusion of Recent Vulnerabilities:** The selected VA datasets have been recently updated and maintained, and the latest vulnerabilities have been included in these datasets.
- **Criterion-2: Number of Vulnerabilities:** The vulnerabilities in the VA dataset cover a large number of projects, and the number of vulnerabilities is considerable. For example, VDISC covers the maximum number of projects (over 1K distinct projects) with 1.2M unique vulnerabilities [76].
- **Criterion-3: Usage in Security Communities:** The selected VA datasets contain multiple programming languages and have been widely used in security communities. For example, the KB, BigVul, and Debian datasets are referenced by over 100 works as benchmark datasets.

In the path C_t at timestamp t, we generate plain texts to describe these paths based on reasoning. For example, the generated **text description** of path (O_1, A_1, O_2) is "***from the observation*** O_1***, we ask LLM to take the action*** A_1***, and the next operation is*** O_2". Then, we aim to retrieve the golden knowledge to correct the factual errors with TF-IDF [81] text similarity. The TF-IDF utilizes the term frequency to calculate the text similarities, which has high efficiency and can focus on vulnerability-related keywords in similarity calculation, e.g., "XSS" and "CSRF", etc. Based on these five selected VA datasets, we extract the golden knowledge $Know_t$ in these datasets that have TF-IDF similarities higher than the threshold θ_{sim}:

$$Know_t | sim_{Know_t \in \mathcal{D}}(Know_t, C_t) > \theta_{sim}, \text{where TF-IDF} \rightarrow sim(\cdot, \cdot) \tag{3}$$

where $sim(\cdot, \cdot)$ is the function for TF-IDF similarity. The TF-IDF is based on the term frequency in the texts, so we calculate the term frequency in all the rich-text information in IRs. All screenshots are parsed to text with OCR.

Since we utilize the TF-IDF similarity to retrieve the external knowledge from the VA dataset, it might introduce some noisy data that cannot be treated as the golden knowledge for the corresponding IR. To address it, we carefully tune the parameter θ_{sim}'s value from $[0.0, 1.0]$ to make sure that all the IRs with factual errors can find the matched golden knowledge in the retrieved knowledge in $Know_t \in \mathcal{D}$. The LLMs can utilize their reasoning ability to analyze the retrieved $Know_t$ and correct the errors in the nodes and edges of reasoning graphs (details of this threat and its alleviation are shown in Section 6.5). We concatenate knowledge and path $Know_t \oplus C_t$ and feed it to the LLM. Taking Fig. 1 as an example, the prompt of the factual error correction is involved in the reasoning graph generation, and we have illustrated an example prompt to guide the reasoning graph correction by using OWASP's golden knowledge as follows:

Table 1. The statistics of VA datasets, which are treated as golden knowledge for factual error correction.

Id	VA Dataset	Updated	Lang	Size
$\mathcal{D}_1$	KB [76]	2023	All	1.2K
$\mathcal{D}_2$	BigVul [33]	2023	C/C++	4.6K
$\mathcal{D}_3$	OWASP [68]	2024	C#/Python	21K
$\mathcal{D}_4$	Debian [11]	2024	C/C++	3.3K
$\mathcal{D}_5$	VDISC [80]	2024	C/C++/Java	1.2M

$P_{correct}$: Prompt for Correcting Factual Error

 # Vulnerability-related IR for Graph Generation
- Input: *The plain text of the historical IR.*
- Vulnerability Type: *The type of the vulnerability (CWE-79).*
 # Path to be Corrected
- Input Path: *From the observation O_1, we ask LLM to take the action $A_{1.4}$ (Analyze* `[SRC4]`*), then we get the observation $O_{2.4}$ (The JS logic injection that causes the XSS payload generation). After that, LLM terminates the agent, and the $O_{3.2}$ shows that CWE-79 is identified.*
 # Example for Correcting Factual Error
- (OWAP Top-10 A03→PortSwigger: Server-side template injection) *... Identify the type of template engine being used. Review its documentation for basic syntax, security considerations, and built-in methods and variables. Explore the template environment and map the attack surface. Audit every exposed object and method, ...)*

Based on the previous prompt, each path can be corrected by comparing with the golden knowledge. After these previous three steps, we output the vulnerability reasoning database *Reason_Base*, where each record is the $\mathcal{G} \in$ *Reason_Base*.

3.2 Retrieving the Relevant Reasoning Graphs to Target IR

As illustrated in the previous sections, it is time-consuming to re-conduct the reasoning to identify vulnerability-related target IRs. Therefore, we introduce the thought of RAG to retrieve the necessary knowledge and generate guidance for identifying the target IR's vulnerability. This incorporates two steps to improve the quality of the generated guidance prompt, i.e., external knowledge retrieval and text generation. Previous work indicates that the generation results of RAG are diversified based on these two steps [45]. However, existing RAG approaches cannot be directly applied to retrieve relevant reasoning graphs from the database, due to the following challenges:

Challenge-1: Heavy time cost in reasoning graph retrieval. In Fig. 3's example, we can see that the reasoning graph contains massive exploration paths, and directly sampling the relevant records based on the complete graph may have massive time and memory costs. We have applied some previous RAG approaches to this task, such as AutoRAG [47], REALM [37], and MemoRAG [77], etc. (note that we investigate the RAGs that have open-sourced datasets and models, and released mature tools), need to take ≥ 5 min/per target IR to retrieve the relevant graphs, which cannot be directly applied in the reasoning graph retrieval.

Algorithm 1: Process of $\mathcal{G}$'s RW.

Input: The target IR $TarIR$; reasoning graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{O_1, O_2, ..., O_n\}$, $\mathcal{E} = \{A_1, A_2, ..., A_n\}$.
Output: The reserved reasoning graph after pruning $\mathcal{G}_r = (\mathcal{V}_r, \mathcal{E}_r)$, where $\mathcal{V}_r \subset \mathcal{V}$, $\mathcal{E}_r \subset \mathcal{E}$.

1 $\tilde{M} = 0$; // Initialize the adjacency matrix.
2 **for** $O_i \in \mathcal{V}, O_j \in \mathcal{V}$, *and* $O_i \xrightarrow{A_i} O_j$ **do**
3 $\tilde{M}_{i,j} = sim(O_j, TarIR) - sim(O_i, TarIR)$, where TF-IDF→ $sim(\cdot,\cdot)$; // (1) Construct the adjacency matrix $\tilde{M}_{i,j}$ of the reasoning graph $\mathcal{G}$.
4 $deg(O_i) = \sum_{O_j \in \mathcal{V}} \tilde{M}_{i,j}$, $deg(O_j) = \sum_{O_i \in \mathcal{V}} \tilde{M}_{j,i}$; // (2) Calculate the sampling probability.
5 $p(O_i, O_j) = \left(\frac{1}{deg(O_i)} + \frac{1}{deg(O_j)}\right) / \sum_{(O_i', O_j')} \left(\frac{1}{deg(O_i')} + \frac{1}{deg(O_j')}\right)$;
6 **end**
7 $\mathcal{V}_r \cup \{O_1\}$, $\mathcal{V}_{walk} = \{O_1\}$; // (3) Random-walking with sampling probability.
8 **for** $O_i \in \mathcal{V}_r \cup \{O_1\} \backslash \mathcal{V}_{walk}$ *and* $O_j \in \mathcal{V} \backslash \mathcal{V}_r$ **do**
9 Select O_j for the node O_i with the probability $p(O_i, O_j)$;
10 $\mathcal{V}_{walk} \cup \{O_i\}$, $\mathcal{V}_r \cup \{O_j\}$, $\mathcal{E}_r \cup \{A_i | O_i \xrightarrow{A_i} O_j\}$;
11 **if** $A_i = AgentTerminator$ **then**
12 **break**; // Loop terminates.
13 **end**
14 **end**
15 return $\mathcal{G}_r$;

Challenge-2: Low relevance in retrieved graphs after pruning. To improve the efficiency of the graph retrieval, we need to prune the reasoning graphs and find a short path to indicate the vulnerability triggering. However, the vulnerability triggering logic is quite complex, and the traditional methods to prune the reasoning graphs, such as DFS [78], CART [57], and PageRank [70], lack the background knowledge of the vulnerability triggering. Even though these graph pruning methods traverse the whole reasoning graph to determine what nodes and edges will be pruned, the retrieved graphs may not be relevant to the target IRs.

To address these two challenges, we introduce the **Random Walking (RW)** [74] method to prune the reasoning graph, which randomly selects paths to represent the $\mathcal{G}$ based on the sampling probabilities, thus reducing the cost of relevant record sampling. We introduce two TF-IDF similarities in the record sampling. (1) The first similarity constructs the weights that measure the similarity between the rich-text information in target IR and the texts of $\mathcal{G}$'s nodes, and the sampling probabilities are calculated with these weights. Nodes with higher probabilities are more likely to be reserved. (2) The second similarity selects the relevant graphs by calculating the similarity between the target IRs and the text description of pruned $\mathcal{G}$.

3.2.1 RW-based Reasoning Graph Pruning. Compared with the traditional graph pruning methods [57, 70], RW does not require the traversal of the whole reasoning graph, which has high efficiency in the relevant record sampling. The main idea of the RW is to estimate the edge weights in the complex graph $\mathcal{G}$ based on some pre-defined criteria (in this work, the estimation criterion is the similarity difference between the target IRs and the nodes in the reasoning graph), which is stored in the Algorithm 1's adjacency matrix $\tilde{M}$. Then, the RW only reserves a few representative nodes and edges based on edge weights, where we conduct the following retrieval process with these nodes.

We propose the Algorithm 1 to reserve the $\mathcal{G}_r = (\mathcal{V}_r, \mathcal{E}_r)$ to represent the reasoning graphs. In this algorithm, from line 1 to 3, we construct the adjacency matrix $\tilde{M}$ of the graph with the target IR, where $\tilde{M}_{i,j}$ measures the importance of the reasoning $O_i \xrightarrow{A_i} O_j$. We utilize the increment of TF-IDF similarity to measure the importance of reasoning. From line 4 to 5, we calculate the sampling probability $p(O_i, O_j)$ with the GraphSaint

edge probability [107]. This probability is calculated with the node degrees, which have unbiasedness in random-walking. From line 7 to 14, we random walk the $\mathcal{G}$ with the sampling probability $p(O_i, O_j)$. We first sample all observations adjacent to O_1, and then we sample other nodes based on the visited observations until the action is terminated. All nodes and edges will not be repeatedly explored. Finally, we reserve the graph $\mathcal{G}_r$ after the random-walking algorithm.

3.2.2 Reserved Graph's Text Description Generation. For each record in the vulnerability reasoning database, we first prune the graph and reserve the subgraph $\mathcal{G}_r$. Second, we extract the paths from $\mathcal{G}_r$ that can achieve the termination. For example, $\mathcal{G}_r$ in Fig. 3 has two paths: $(O_1, A_{1.1}, O_{2.1}, A_{2.1}, O_{3.1})$ and $(O_1, A_{1.4}, O_{2.4}, A_{2.4}, O_{3.2})$. The textual description of $\mathcal{G}_r$ is the concatenation of all the paths' textual descriptions, as is described in Section 3.1.3.

3.2.3 Relevant Reasoning Graph Retrieval. Finally, we utilize the TF-IDF text similarity to select the relevant graphs with values higher than the threshold θ_{sim}.

$$\mathcal{G}_r^{Tar} | sim_{\mathcal{G}_r^{Tar} \in Reason\_Base}(TarIR, \mathcal{G}_r^{Tar}) > \theta_{sim}, \text{where TF-IDF} \rightarrow sim(\cdot, \cdot) \tag{4}$$

where $\mathcal{G}_r^{Tar}$ is the textual description of selected reasoning graphs, and $TarIR$ indicates all the rich-text information in target IRs, which are parsed to text by OCR and CAST tools.

3.3 Identifying the Vulnerability of Target IR

Since manually designed prompts are not appropriate for some specific tasks, researchers utilize LLM to generate specialized guidance prompts, thus improving their performances [112]. Given the target IR and its relevant reasoning graphs $\mathcal{G}_r^{Tar}$, we utilize the LLM to generate guidance prompts for different target IRs. We ask the generated guidance prompt $(\mathcal{G}_r^{Tar}, TarIR) \mapsto P_{guide}$ to include several steps, which describe the instructions for how to analyze target IR. Fig. 3 shows the example of the guidance prompt that includes the three steps to analyze the rich-text information in the target IR. In these steps, the guidance prompt refers to the reserved graph. It finds the similarity between the target IR and relevant record in page redirection logic and asks the LLM to analyze the main page and the XSS-triggered page to identify vulnerabilities.

After generating the guidance prompt, we concatenate it with the prompt for identifying vulnerability $P_{identify}$. The following is the concatenated prompt $P_{guide} \oplus P_{identify}$:

Concatenated Prompt for Vulnerability Identification

 # Identification of Vulnerability

$P_{identify}$: Please identify whether the following IR contains the vulnerability, and predict the type (CWE-ID) of the vulnerability. *This is a classification task, so please directly output whether the IR contains the vulnerability with "Yes, No".* The output format for vulnerability identification is {Yes, No}. Moreover, you need to just directly output the {CWE-ID} without other information.

 # Guidance to Identify Vulnerability

P_{guide}: According to the relevant reasoning graph, the generated guidance prompt contains the following steps. We will concatenate it with the $P_{identify}$: (Several steps STEP-1~STEP-*n*, where each step contains the instruction for how to analyze the target IR).

- ***Input:*** *The content of target IR, which is formatted as JSON.*
- ***Input:*** *The textual description of all the selected graphs* $\mathcal{G}_r^{Tar}$.

where the prompt requires the output to explicitly output the classification results of vulnerability identification with two labels "Yes (1), No (0)", as well as the type of vulnerability "CWE-ID". For vulnerability identification, we use the LLM's API to output the **probability** (e.g., ChatGPT utilizes the attribute in the response `logprobs.top_logprobs` to calculate each output label) rather than the labels.

Referring to the previous works [72, 82, 86, 102], using the probability to determine the vulnerability-related IRs is a flexible method when the testing dataset is extremely imbalanced, and the output probability of positive label "Yes" to identify the vulnerability, i.e., $p(\text{``}Yes\text{''}|TarIR) \in [0.0, 1.0]$. We utilize the threshold $\theta_{out}$ to decide the classification result, and $p(\text{``}Yes\text{''}|TarIR) \geq \theta_{out}$ means that the target IR is the vulnerability-related IR. For CWE-ID prediction, we output the CWE-ID rather than using the classification probability, because the labels have similar distributions in the dataset. We feed this concatenated prompt target IR to the LLM and identify its vulnerability. Since a vulnerability-related IR may have vulnerabilities with multiple CWE-IDs, we will refine the dataset by splitting them into multiple IRs (Section 4.1).

4 Experimental Design

To evaluate the performance of VulRTex, we investigate the following research questions (RQs):

- **RQ1 (Performance of VulRTex):** How does VulRTex perform on identifying the vulnerability-related IR and predicting the CWE-ID?
- **RQ2 (Ablation Study):** How does each component contribute to identifying vulnerability-related IRs and predicting CWE-IDs?
- **RQ3 (Practical Application):** Can VulRTex identify emerging vulnerabilities from the open-source projects?

4.1 Dataset Preparation

In this section, we prepare our dataset in the following four steps: First, we select and collect the dataset after a rigorous review and selection process with security practitioners. Second, our manual analysis illustrates that some IRs may contain vulnerabilities that correspond to multiple CWE-IDs, so we refined the dataset by re-annotating the CWE-IDs to improve the data quality. Third, we collect the original IR to improve the dataset with its rich-text information. Finally, we preprocess the dataset with token replacement.

STEP-1: Selecting & Collecting the Dataset. Before the dataset preprocessing, we search for five candidate data sources, i.e., **GHArchive** [8], **D2A** [111], **VulZoo** [79], and MITRE's **ATT&CK** [59] & **CAPEC** [29]. Since the selected data sources for evaluating the VulRTex are large-scale and should contain traceable IRs, the criteria for selecting the data source will not be limited to selecting VA datasets (Section 3.1.3), i.e., **Inclusion of Recent Vulnerabilities (IRV)**, **Number of Vulnerabilities (NV)**, and **Usage in Security Communities (USC)**. We strictly follow Wu's work on vulnerability database reviewing [100] and add three additional criteria to review the evaluation dataset, and the selected data sources should satisfy all the criteria after reviewing.

Table 2. The results of reviewing the data sources based on VA/evaluation dataset criteria.

Data Source	VA-Dataset Criteria			Eval-Dataset Criteria		
	IRV	NV	USC	TIR	CPL	NNS
GHArchive [8]	✔	✔	✔	✔	✔	✔
D2A [111]	✘	✘	✔	✔	✔	✘
ATT&CK [59]	✘	✔	✔	✘	✔	✔
VulZoo [79]	✔	✔	✘	✔	✔	✘
CAPEC [29]	✔	✘	✔	✔	✘	✔
KB [76]	✔	✔	✔	✘	✔	✔
BigVul [33]	✔	✔	✔	✘	✘	✔
OWASP [68]	✔	✔	✔	✘	✘	✔
Debian [11]	✔	✔	✔	✘	✘	✔
VDISC [80]	✔	✔	✔	✘	✘	✔

- **Criterion-1: Traceability of IRs (TIR).** The original vulnerability-related IRs will be traceable based on each sample's details, and the CWE-IDs are also traceable in the data sources.
- **Criterion-2: Coverage of Programming Languages (CPL).** The vulnerabilities in the dataset include various programming languages, and VulRTex cannot be limited to identifying vulnerabilities in only a few programming languages.
- **Criterion-3: Number of Noise Samples (NNS).** The data sources should not contain too much noise, including incorrect labels of vulnerability and CWE-ID.

From Table 2, we can see that only the **GHArchive** satisfies all the criteria after dataset review and can be used as the data source for vulnerability identification. Note that, these data sources are different from the previous VA dataset After we determine the data source we use in the experiment, we collect the dataset by strictly following Pan et al's work's basic process of constructing the dataset [72], so the labels of whether the IRs contain the vulnerability (i.e., **the first task:** identifying the vulnerability-related IRs, where the label is "Yes or No") are correct, and the dataset is considered as a benchmark dataset without noise data (i.e., IRs that are mislabeled with incorrect labels). It is a comprehensive dataset that contains 3,884 officially disclosed vulnerabilities with their source links of GitHub IRs since 2015. This dataset is manually labeled by security practitioners in CVE and has been widely used in vulnerability identification tasks [27, 50, 52, 65, 72, 80]. We collect the vulnerability information in this dataset, such as whether the IR is vulnerability-related, and its disclosed CVE/CWE-ID.

STEP-2: Refining the Dataset. Since the labels of vulnerability-related IR identification are reviewed and double-checked regularly by experienced security practitioners from well-known security organizations/institutions (e.g., CVE, NVD, OWASP, etc.), we believe that these labels are correct. However, for the labels of CWE-ID (i.e., **the second task:** predicting the CWE-IDs of the vulnerability) in our manual analysis, around 118 of 3,884 vulnerability-related IRs may contain vulnerabilities that correspond to multiple CWE-IDs. The original GHArchive dataset may miss these CWE-ID labels, which affects the reliability of evaluation results. For example, the labeled CWE-ID in *Fuel-CMS/issues/536* is the "XSS Injection" (CWE-79) [61]. Still, we find that the project may also encounter the "Cross-Site Request Forgery (CSRF)" (CWE-352) [60] based on the description of this IR, but the CWE-352 label is missing in the dataset. To improve the quality of our dataset, we refine the dataset by adding new IRs with the missed labels. To reduce the biases in the manual re-annotation, we have invited three security practitioners who are not among the authors, who have over five years of experience in software security, to determine whether the re-annotated dataset is correct. We ask them to independently check whether the new samples are accurate. The average Cohen's Kappa [73] value is 0.9, which achieves a high agreement on the labels of the reannotated dataset.

STEP-3: Collecting the Original Rich-text Information. For all the samples in the dataset, we retrieve their original rich-text information. For GHArchive, each sample contains an external link to the original address of the GitHub IR. We search the source of all the GitHub IRs and utilize Python's `BeautifulSoup` package to crawl all the HTML elements in the original pages, including the links to page screenshots (wrapped by `<a href=".jpg|.png">`, `</a>`) and code snippets (wrapped by `<code>`, `</code>`).

STEP-4: Preprocessing the Dataset. The GitHub IRs collected from the web pages are in HTML format, so we preprocess the IRs with the specific token tagging and data merging/splitting.

For token tagging, (1) we use ***[SCR]*** to tag the screenshots and ***[CODE]*** to tag the code snippets. Then, we add new JSON fields to store the content of rich-text information in the input IRs so that each IR can be represented as {"Content": "***Text***&***[SCR]***&***[CODE]***", "Rich-Text": ["***[SCR]***": "Link of Screenshot", "***[CODE]***": "Detail of Code Snippet"]}; (2) we search for the privacy information in the IRs, such as the personal identifiers (e.g., "*username*", "*password*", and "*email address*", etc.) and project identifiers (e.g, "*project source files*" and "*project configuration files*", etc.). We have tagged the privacy elements with the symbol ***[PIV_{InfoName}]***, where ***{InfoName}*** is the type of these elements. To improve the coverage of privacy information, we ask the security practitioners to annotate and discuss in multiple turns; and (3) we remove other HTML tags (e.g., `<td>`, `<tr>`, `<p>`, etc.) and retain the plain text inside these tags, then correct typos and lemmatize the texts [85];

For data merging/splitting, (1) we merge similar code snippets and page screenshots, which may have a few differences and describe similar vulnerability-related information; and (2) we split the vulnerability-related IRs by sorting the IRs in time order, then choose the first 60% of the IRs as the historical IR and the remaining 40% as the target IR (proportion setting is decided by the hyper-parameter tuning, which is shown in Section 6.1).

Table 3 shows the number of IRs in our dataset, where the column **#Total** and **#R-Text** indicate the total number of IRs, and IRs with rich-text, and the number **#Total-#R-Text** illustrates the IRs with only plain text.

Table 3. The statistics of the number of IRs in the original and refined datasets.

Dataset		Original				Refined*			
		#Total	#R-Text	#SCR	#CODE	#Total	#R-Text	#SCR	#CODE
Vul-IR	Historical	2,306	683	495	619	2,401 (+95)	720 (+37)	527 (+32)	630 (+11)
	Target	1,578	956	664	760	1,601 (+23)	966 (+10)	670 (+6)	767 (+7)
Non-Vul-IR	Historical	714,790	466,179	352,675	330,753	714,790	466,179	352,675	330,753
	Target	476,528	201,391	139,578	276,639	476,528	201,391	139,578	276,639

* The refined dataset for training and evaluating VulRTex and baselines. The value after each IR's number, i.e., (+*value*), indicates the increased number of IRs after dataset refining (**STEP-2**). Note that, the dataset refining step only adds the missing CWE-IDs for the vulnerability-related IRs, so the numbers in the **Non-Vul-IR** will not be changed.

Column **#SCR**, and **#CODE** indicate the number of IRs with page screenshots and code snippets in the **#R-Text**. The labels **Vul-IR** and **Non-Vul-IR** indicate whether the IR is vulnerability-related. We obtained 1,686/4,002 rich-text vulnerability-related IRs, where 2,401 (60%) of them are historical IRs, and 1,601 (40%) are target IRs.

4.2 Baselines

To evaluate the performance of our approach, we find that identifying vulnerabilities from IRs lacks baselines, so we manually select approaches that can be used in the vulnerability detection and natural language process (NLP) tasks, and then we select the baselines based on the following criteria:

- **Criterion-1: Baselines' Applicability.** The selected baselines can be applied to our tasks. For example, traditional vulnerability detectors mainly focus on finding vulnerabilities from source code in the repository, which cannot be applied to identify vulnerabilities from IRs.
- **Criterion-2: Baseline Artifacts' Availability.** Considering the reliability of the baseline's result, we select the baselines whose dataset and code are available. For the approaches for which their artifacts are not available, we have asked the authors of these approaches and conducted the experiments on the models if the authors respond to us.
- **Criterion-3: Baselines' Performance & Efficiency.** With the development of LLMs, we need to select the models that are adaptive to the vulnerability identification tasks. After discussing with the security practitioners who participate in our dataset refining and following the previous work [38], we have specified the criterion for choosing baselines that have better performance and lower time cost than the other models: For DL baselines, we choose them whose performances are $\geq$ 30% F1 on average (DL baselines in MemVul, e.g., Random Guess, Random Forest, and Naive Bayes, are not chosen). For the LLMs, we choose the models whose time costs of prediction are $\leq$ 2 minutes/IR. Based on this criterion, the GPT-o1 (2.5 minutes/IR) LLM reasoning and RAG approaches, e.g., AutoRAG (2.2 minutes/IR), MemoRAG (2.6 minutes/IR), and ToolBench (4 minutes/IR), are not chosen, because they may not be useful in practice.
- **Criterion-4: API calling fees.** Some black-box LLMs provide paid APIs and more efficient model versions. However, we find that for some models like GPT-o1, the cost of identifying vulnerabilities even reaches $10 per rich-text IR. Considering the public welfare and open-source nature of our method, we plan to evaluate the LLM's performance on free or low-cost LLMs, so we do not choose GPT-o1 with high API calling fees as the baseline in our experiment.

After evaluating the novel approaches in vulnerability identification, LLM's reasoning, and RAG, we finally select three types of baselines that meet the previous three criteria, i.e., three deep learning (DL) and LLM baselines that are widely used in NLP tasks, with three LLM reasoning strategies that are applicable in identifying vulnerability-related IRs and predicting CWE-IDs.

DL Baselines. We first retrain the Deep Learning (DL) baselines on our dataset to identify the vulnerability-related IRs and predict their CWE-IDs. **LR** [54] is the linear regression method that predicts the vulnerability with the single-layer perception; **MLP** [49, 99] utilizes the multi-layer perceptions to classify the type of vulnerability. In our experiment, these two DL baselines can outperform other baselines in identifying vulnerabilities from rich-text information. Also, we introduce **MemVul** [72], which is the latest approach that utilizes the memory network to store the relationships between vulnerability types and reuse it to identify the new IRs.
LLM Baselines. In addition to DL baselines, we also compare the VulRTex with the original LLMs, since they are widely used and achieve high performances on text classification and generation tasks. These LLM baselines use the same prompt as VulRTex, i.e., $P_{identify}$, in Section 3.3 to identify the vulnerability by outputting the confidence scores. Considering the hardware limitation and the model usage frequency in the industry, we have chosen three types of well-maintained LLM baselines, which are shown as follows:

- **Text Generative LLMs:** These three baseline is pre-trained on plain text-pairs, which aim to improve the ability to analyze complex user queries. **LLaMA** [91] is the LLM proposed by Meta and is trained on multiple language models with various inference budgets; **GPT-3** [105] and **GPT-3.5** [66] are representative text-generative LLMs proposed by OpenAI, which use over 100B of parameters and are trained on over 10TB samples with multiple training strategies (few-shot, zero-shot, etc.). We choose the following stable versions of LLMs: LLaMA (*Llama-2-13b-chat-hf*), GPT-3 (*text-davinci-003*), and **GPT-3.5** (*gpt-3.5-turbo*).
- **Multimodal LLMs (MLLMs):** The MLLMs are pre-trained on multimodal data, where we input all the rich-text information into the MLLMs into these models to predict the CWE-IDS. **Qwen2VL** [103] is proposed by Alibaba, which is a well-maintained MLLM that can analyze both image and text queries. Limited by the hardware resources, we chose the *qwen-2vl-7b-instruct* version in this task. **GPT-4o** [67] is the MLLM proposed by OpenAI that can effectively understand visual queries' semantic information.
- **Post-Trained LLMs:** To make the output of LLMs in line with human cognition, researchers have proposed the post-trained alignment to improve the LLMs. **DeepSeek-R1** [31] is a well-maintained LLM that introduces Reinforcement Learning from Human Feedback (RLHF) in the long-term reasoning. We have compared GPT-o1 with DeepSeek-R1, which has a heavy reasoning time cost and large API fees, and the vulnerability identification results are less than those of DeepSeek-R1 (-5.0% F1 on average).

LLM Reasoning Strategies. Since LLMs are well-trained in these versions, we will not retrain them on our dataset. Moreover, for each LLM, we also introduce three reasoning strategies to analyze the rich-text information, i.e., **+CoT-SC** [96], and **+ReAct** [104], which are the latest frameworks that control the LLM's reasoning with the highest performances on our tasks. CoT-SC utilizes the majority-voting technique to decide the reasoning step with the highest confidence, and ReAct adopts the "*Thinking-Observation-Action*" framework in the reasoning. **+DS-Agent** [36] is the latest approach that firstly combines the RAG in the LLM reasoning process.

To ensure a fair comparison, all the DL and LLM baselines use the same JSON-formatted IRs as input datasets with rich-text information, as is illustrated in **STEP-4** of Section 4.1.

4.3 Evaluation Metrics

Evaluation Metrics on Identifying Vulnerability-related IR. Since the labels of target IRs are imbalanced in Table 3, we choose two sets of appropriate metrics to measure the performance of vulnerability-related IR identification. The first set of metrics measures the performance of VulRTex on identifying the positive vulnerability-related IRs, i.e., **Precision**, **Recall**, and **F1-Score**. These three metrics are useful in text classification tasks when the labels in the testing dataset are imbalanced. Precision calculates the ratio of correct positive predictions to the total positive predictions; Recall calculates the ratio of correct positive predictions to the ground-truth positive labels; and F1-Score is the harmony of Precision and Recall.

Besides, since the number of negative samples is much larger than the positive ones (*imbalanced dataset*), we choose threshold-independent metrics to measure the performances of VulRTex when the dataset is extremely imbalanced, i.e., **AUROC** (area under the Receiver Operating Characteristics curve) and **AUPRC** (area under the Precision-Recall curve). To intuitively illustrate the benefits of VulRTex, we plot the Precision-Recall curves by tuning the threshold θ_{out} within $[0.0, 1.0]$ with 0.5 as the interval and measure the trade-off values. Previous works illustrate that AUROC and AUPRC are indicative metrics when the dataset imbalance affects VulRTex's performances [30, 88].

Evaluation Metrics on Predicting CWE-IDs. To measure the performance in CWE-ID prediction, we apply the **Macro-P**, **Macro-R**, and **Macro-F1**. Macro-P and Macro-R are the macro averages of precision and recall on all the CWE labels (i.e., the equations are $\frac{1}{n}\sum_{cwe_i}^{n} pre(cwe_i)$ and $\frac{1}{n}\sum_{cwe_i}^{n} rec(cwe_i)$), and Macro-F1 is the harmony of Macro-P and Macro-R. Due to the equal importance of all the CWE-IDs, the macro average value can better reflect the prediction results than the micro average value. Since we only evaluate CWE-ID prediction with positive labels in vulnerability identification, and different CWE-IDs have similar distributions in the vulnerability-related IRs, we will not utilize the AUROC and AUPRC in the CWE-ID prediction task.

Evaluation Metrics on Time Cost. To compare the time cost of the baselines and VulRTex, we utilize the average seconds of identifying a vulnerability-related IR and predicting its CWE-ID, which is denoted by s/IR. We calculate the average time cost after all the experiments are conducted.

4.4 Experimental Settings

Experiment Details. Since the LLM's output may have randomness, we conducted **20-time experiments** on target IRs and calculated the mean value of these evaluation results, i.e., $\frac{1}{20}\sum_{i}^{20} res_i$, to reduce the biases, where res_i indicates the i_{th} metric result of experiments. We use the LLM's API, such as `logprobs.top_logprobs` to output the probability and `message.content` to output the response text. To evaluate the advantages and practical application, we conduct an ablation study on four types of components and apply VulRTex on the newly proposed IRs to observe the proportion of disclosed vulnerabilities.

Hyper-parameters. We first set the proportion of the historical IRs as 60% and the threshold $\theta_{sim} = 0.7$ (parameter settings are shown in Section 6.1). Then, we set the threshold for determining the output of vulnerability, i.e., θ_{out}, within the range $[0.0, 1.0]$, and choose the optimal F1 value when $\theta_{out} = 0.55$ (the tuning of the θ_{out} may affect the trade-off between Precision and Recall, so we illustrate the curve in Section 5). For each value in the parameter tuning, we also conduct 20-time experiments to determine whether the output of DL/LLM will not have biases and calculate the average score of the repeated experiments. For other parameters in VulRTex's LLM module and the baselines, we set the value of them in DL/LLM, such as the number of layers, learning rate (i.e., $2e^{-5}$ for BERT encoder and $1e^{-4}$ for other modules), temperature (i.e., 0.0 for all the LLMs to ensure a stable probability distribution), etc., as default values.

Hardware. All the baselines and variants are retrained on these IRs in the refined dataset and run on a high-performance server with Ubuntu OS, NVIDIA RTX A6000 GPUs, and 64GB RAM.

5 Results

5.1 RQ1: Performances of Identifying Vulnerability-related IRs and Predicting CWE-IDs

In this experiment, we first evaluate the performances of VulRTex on the overall target IRs in our dataset, Then, we analyze the performances of VulRTex on target IRs with/without rich-text information and compare the performance of VulRTex with the best-performed DL/LLM baselines, i.e., VulRTex and GPT-3.5. We also plot the Precision-Recall (PR) curves to intuitively illustrate the advantages of VulRTex to the baselines on different output thresholds θ_{out}.

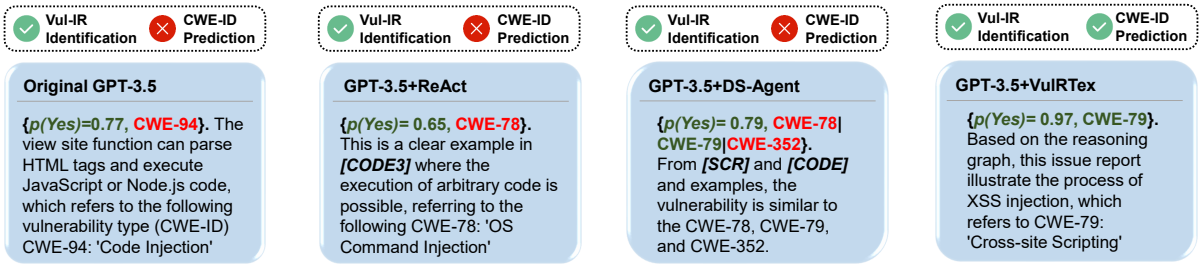
Table 4. The results of baseline comparison of VulRTex on identifying vulnerability-related IRs and predicting CWE-IDs of these vulnerabilities on overall target IRs (%).

Category	Methods	Time Cost (s/IR)	Vul-IR Identification (VI, %)					CWE-ID Prediction (CP, %)		
			Precision	Recall	F1-Score	AUROC	AUPRC	Macro-P	Macro-R	Macro-F1
DL Baselines	LR	1.0	43.7	24.0	31.0	90.3	30.5	34.1	31.7	32.9
	MLP	1.7	35.2	34.7	34.9	91.9	19.6	42.7	45.8	44.2
	MemVul	2.0	38.3	71.6	49.9	96.5	33.2	46.5	52.4	49.3
Performances of Text Generative LLMs										
LLaMA Baselines & Our Approach	Original	**3.6**	66.5	74.1	70.1	92.6	35.0	57.5	66.7	61.8
	+CoT-SC	11.7	68.2	70.3	69.2	94.0	39.6	59.4	69.2	63.9
	+ReAct	15.0	69.9	70.1	70.0	94.1	40.2	60.2	65.7	62.8
	+DS-Agent	20.0	69.3	69.5	69.4	93.6	37.9	62.3	61.7	62.0
	+VulRTex	7.9 (↑ 4.3)	**74.5** (↑4.6)	**75.3** (↑1.2)	**74.9** (↑4.8)	**97.9** (↑3.8)	**65.9** (↑25.7)	**68.5** (↑6.2)	**72.4** (↑3.2)	**70.4** (↑6.5)
GPT-3 Baselines & Our Approach	Original	**4.9**	50.8	72.0	59.6	96.0	39.6	66.8	60.3	63.4
	+CoT-SC	19.5	62.5	76.4	68.8	96.9	41.5	67.2	61.5	64.2
	+ReAct	21.0	63.0	69.7	66.2	96.7	45.5	67.5	62.0	64.6
	+DS-Agent	25.5	64.6	98.5	47.7	96.1	39.9	63.9	67.9	65.8
	+VulRTex	9.2 (↑ 4.3)	**69.5** (↑4.9)	**77.0** (↑0.6)	**73.1** (↑4.3)	**98.2** (↑1.3)	**77.2** (↑31.7)	**69.4** (↑1.5)	**75.3** (↑7.4)	**72.2** (↑6.4)
GPT-3.5 Baselines & Our Approach	Original	**4.2**	80.7	66.3	72.8	97.3	45.9	76.2	63.9	69.5
	+CoT-SC	26.9	85.3	70.2	77.0	98.1	46.8	71.3	70.2	70.7
	+ReAct	30.0	86.9	67.4	75.9	98.5	53.6	72.5	70.7	71.6
	+DS-Agent	44.5	84.2	72.5	77.9	70.2	60.2	75.3	73.2	66.7
	+VulRTex	11.7 (↑ 7.5)	**90.2** (↑3.3)	**87.7** (↑15.2)	**88.9** (↑11.0)	**99.2** (↑0.7)	**80.4** (↑20.2)	**82.5** (↑6.3)	**85.0** (↑9.7)	**83.7** (↑10.5)
Performances of MLLMs										
Qwen2VL Baselines & Our Approach	Origin	**10.6**	89.3	56.2	69.0	94.2	52.5	67.7	61.9	64.7
	+CoT-SC	12.3*	86.1	59.0	70.0	92.5	50.3	70.4	69.2	69.8
	+ReAct	62.5	87.5	60.3	71.4	95.1	51.5	72.9	68.7	70.7
	+DS-Agent	57.9	88.4	61.5	72.5	95.5	70.6	80.1	74.1	77.0
	+VulRTex	15.9 (↑ 5.3)	**92.9** (↑3.6)	**89.5** (↑28.0)	**91.2** (↑18.6)	**99.6** (↑4.1)	**85.9** (↑15.3)	**83.4** (↑3.3)	**88.7** (↑14.6)	**86.0** (↑9.0)
GPT-4o Baselines & Our Approach	Origin	**9.6**	92.0	68.0	78.2	92.1	47.6	72.6	79.5	75.9
	+CoT-SC	26.3	94.0	69.7	80.0	94.6	48.3	71.9	77.1	74.4
	+ReAct	77.5	88.7	64.3	74.6	92.4	43.5	74.4	78.2	76.3
	+DS-Agent	84.2	85.1	66.2	74.5	92.1	44.5	72.3	80.3	76.1
	+VulRTex	15.3 (↑ 5.7)	**94.3** (↑0.3)	**83.1** (↑13.4)	**88.3** (↑8.3)	**98.2** (↑3.6)	**88.9** (↑40.6)	**83.4** (↑9.0)	**89.5** (↑9.2)	**86.3** (↑10.0)
Performances of Post-Trained LLMs										
DeepSeek-R1 Baselines & Our Approach	Origin	**31.2**	96.3	80.2	87.5	99.1	83.4	76.8	85.3	80.8
	+CoT-SC	126.9	96.5	84.5	90.1	99.0	84.5	77.1	82.5	79.7
	+ReAct	150.5	97.2	85.1	90.7	**99.3**	84.0	78.3	83.6	80.9
	+DS-Agent	180.7	96.8	89.9	93.2	99.2	89.7	84.1	87.7	85.9
	+VulRTex	33.9 (↑2.7)	**100.0** (↑2.8)	**90.7** (↑0.8)	**95.1** (↑1.9)	99.2 (↓0.1)	**90.2** (↑0.5)	**85.2** (↑1.1)	**90.1** (↑2.4)	**87.6** (↑1.7)

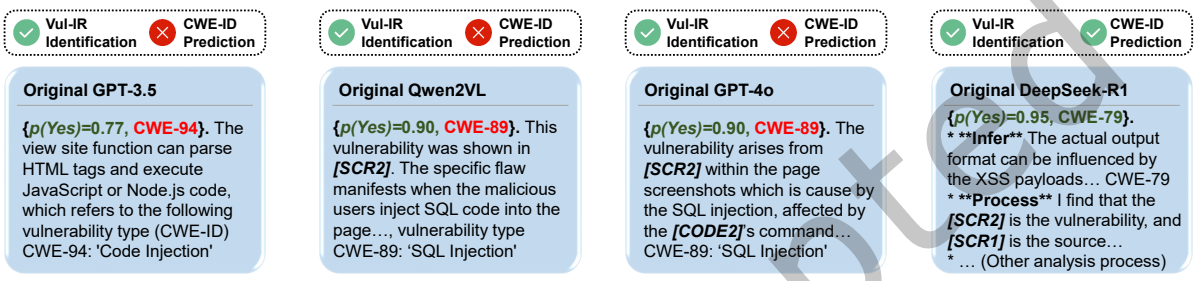
*For Qwen2VL-7B+CoT-SC, it uses the majority voting method in the decision-making process, which stops early for vulnerabilities with high confidence, resulting in less time cost.

Comparison Results among DL/LLM/Reasoning Baselines. Table 4 illustrates the comparison results among different DL, LLM, and reasoning baselines, and the best performance of each column is highlighted with **bold face**. We can see that the VulRTex can outperform all the DL and LLM baselines:

- **Performance Comparison.** The GPT-3.5+VulRTex obtains the highest performances with 88.9% (F1), 99.2% (AUROC), and 80.4% (AUPRC) in vulnerability identification, and 83.7% (Macro-F1) in CWE-ID Prediction. It not only outperforms all the DL baselines, but also outperforms the reasoning baselines, improving the GPT-3.5+DS-Agent (baseline with the highest performance) with +11.0% (F1-Score), +0.7% (AUROC), +20.2% (AUPRC), and +10.5% (Macro-F1).
- **Time Cost Comparison.** The time cost of each LLM+VulRTex is only 5.4 s/IR higher than the original LLMs on average of three selected LLMs, and around 50% lower than the baseline reasoning approaches. Especially



(a) Comparison of VulRTex on different reasoning strategies.



(b) Comparison on different LLMs to GHArchive's vulnerability identification.

Fig. 4. The case study of VulRTex on the motivation example (example in Fig. 1).

for GPT-3.5, the time cost of VulRTex reduces by over -15.2 s/IR. Overall, the trade-off between performance and time cost illustrates the benefits of VulRTex. Moreover, we find that for the DeepSeek-R1 model that requires a long CoT for reasoning, the additional time cost of VulRTex is less than that of other LLMs, which is only 2.7 s/IR higher. Within this time cost, the **Concatenated Prompt-Guided Vulnerability Identification** takes 27.2 s/IR, where **LLM Prompt-based Guidance Prompt Generation** takes 6.6 s/IR. According to the structure of DeepSeek-R1, during its analysis of IRs, this post-training LLM cannot control the long CoT-based reasoning, which may fall into *invalid and repeated exploration*. On the contrary, the generated guidance can help DeepSeek-R1 escape from this repetition, thus improving its efficiency.

- **Case Study.** To qualitatively evaluate the VulRTex, we conduct the **case study** by analyzing the responses of GPT-3.5 on the motivation's IR. We compare VulRTex with the original GPT-3.5 and two state-of-the-art (SOTA) reasoning approaches, i.e., ReAct and DS-Agent. Fig. 4 (a) shows the results of the case study. We can see that the original GPT-3.5/GPT-3.5+ReAct cannot predict the accurate CWE-ID, and GPT-3.5+DS-Agent fails to determine the CWE-ID based on the relevant information it retrieves from the historical IRs. In comparison, VulRTex can accurately predict the CWE-ID based on the reasoning graphs and historical IRs.

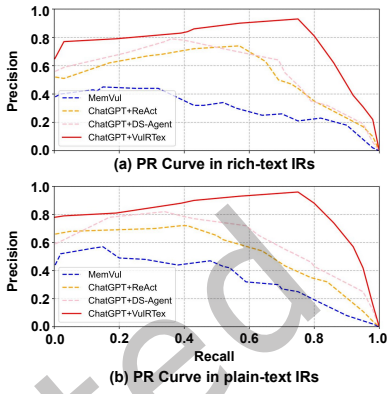
Comparison Results among Different LLM Types. Table 4 also illustrates the performances on MLLMs and Post-Trained LLMs. We also highlight the best performances for each LLM with **bold face**. We can see that VulRTex can outperform most of the LLMs with the reasoning strategies, and we clarify how it improves the two types of LLMs separately:

- **Improvement to MLLMs:** The Qwen2VL and GPT-4o are pre-trained on multimodal data, which can analyze information in the rich-text elements and identify the vulnerabilities with this information. Therefore, we find that the Precision/Macro-P values of these models are higher than those of text-generative LLMs. However,

Table 5. The results of baseline comparison on IR with/without rich-text information on the representative text-generative LLM, i.e., GPT-3.5 (%).

Category	Methods	VI, % F1-Score	VI, % AUROC	VI, % AUPRC	CP, % Macro-F1
IR with Rich-Text	MemVul	37.2	80.6	30.9	36.8
	GPT-3.5	65.3	86.9	34.2	60.8
	+CoT-SC	68.9	89.2	39.0	61.2
	+ReAct	66.2	89.9	53.9	65.3
	+DS-Agent	67.0	87.5	57.5	64.6
	+VulRTex	**85.6** (↑16.7)	**97.9** (↑8.0)	**76.6** (↑19.1)	**88.7** (↑23.4)
IR w/o Rich-Text (Plain-Text)	MemVul	53.2	97.5	36.6	53.7
	GPT-3.5	79.6	98.4	40.2	72.0
	+CoT-SC	80.1	98.9	47.5	75.9
	+ReAct	81.7	99.3	53.4	73.5
	+DS-Agent	82.9	99.1	61.9	74.5
	+VulRTex	**92.2** (↑9.3)	**99.8** (↑0.5)	**81.1** (↑19.2)	**80.2** (↑4.3)

Fig. 5. Precision-Recall (PR) curves of VulRTex and baselines.



MLLMs are sensitive to some error/warning page screenshots or code's testing reports, which lead to false-positive predictions, resulting in lower Recall/Macro-R. VulRTex, on the contrary, alleviates this weakness and improves the performance with over 8.0% (F1) for vulnerability identification and 9.0% (Macro-F1) for CWE-ID prediction.

- **Improvement to Post-Trained LLMs:** The DeepSeek-R1 is optimized by the RLHF method, which can remove some false predictions during the post-training and alignment phase. Therefore, we can see that the Recall and F1 values improve a lot compared to the other LLMs. Comparatively, VulRTex can further improve the vulnerability identification results to DeepSeek-R1, which achieves 100% (Precision), 95.1% (F1), and 87.6% (Macro-F1). Moreover, the time cost of DeepSeek-R1+VulRTex is less than the original model, because the vulnerability reasoning database can accelerate the LLM to analyze the elements in IRs.
- **Case Study:** To intuitively show the results of these LLMs, we also conduct a case study on the output of GPT-3.5, Qwen2VL, GPT-4o, and DeepSeek-R1 (we compare the original models to show their outputs' specific features). Fig. 4 (b) shows the results of the case study. We can see that the two MLLMs (Qwen2VL and GPT-4o) identify the error page screenshot ***[SCR2]***, but they are misled by the pre-trained dataset and incorrectly categorize it as SQL injection (CWE-89). In comparison, DeepSeek-R1 can capture the relationship between ***[SCR1]*** and ***[SCR2]*** with the reasoning and post-training, thus improving the accuracy Based on the aforementioned results, we believe that reasoning is important to vulnerability identification, and our approach can improve the LLM's performance with less time cost.

Comparison Results on IRs with/without Rich-Text Information. Table 5 shows the comparison results on target IR with/without rich-text information. We can see that GPT-3.5+VulRTex significantly outperforms the baselines on both rich-text and plain-text IRs. For rich-text IRs, it significantly outperforms the best baseline by +19.1% (AUPRC) in vulnerability identification and +23.4% (Macro-F1) in CWE-ID prediction. For plain-text IRs, it also outperforms the best baseline by +19.2% (AUPRC) in vulnerability identification and +4.3% (Macro-F1) in CWE-ID prediction.

To further evaluate the ability of VulRTex to adapt to the imbalanced dataset, we plot the **Precision-Recall (PR) curve** of VulRTex and representative baselines, i.e., MemVul, GPT-3.5+ReAct, and GPT-3.5+DS-Agent, which can reflect the performance of these models under different output threshold θ_{out}. In Fig. 5, we can see that the performances of VulRTex are higher than baselines under all the different thresholds, and the area (i.e., AUPRC) under its PR curve surrounds the other baselines' curves, which further illustrates VulRTex's

advantages. Overall, VulRTex outperforms the baselines on both IRs with or without rich-text information, which illustrates its practicality.

Answering to RQ1

VulRTex improves all the baselines, and GPT-3.5+VulRTex outperforms the best baseline with +19.2% (AUPRC) in vulnerability identification and +4.3% (Macro-F1) in CWE-ID prediction, with 2x lower time cost than the LLM reasoning baselines. The PR curves further illustrate its advantages in identifying vulnerabilities in imbalanced datasets. For the MLLMs, VulRTex reduces the sensitivity to the rich-text elements and increases their F1 and Macro-F1 by over 8%. Finally, we find that the post-trained LLM, i.e., DeepSeek-R1, is more effective in the vulnerability identification tasks, where VulRTex can further enhance its capabilities. This inspires us to improve its performance during post-training alignment in the future.

5.2 RQ2: Ablation Study

We have conducted three types of ablation studies to evaluate the components' contributions. First, we compare VulRTex's performances with the following four types of variants, i.e., rich-text information, CoT-based reasoning, reasoning graph retrieval, and text matching similarity. Second, we compare the different OCR tools and illustrate why we select Tencent OCR API for the screenshot parsing. Third, we analyze the contribution of the five selected VA datasets for the factual error correction with a typical output example.

Comparison of Variants on VulRTex's Performances. We have chosen four types of variants that may contribute to the VulRTex's performance, which are summarized as follows:

- **Rich-text Information: w/o *[SCR]*** & **w/o *[CODE]*** (remove page screenshots & code snippets).
- **CoT-based Reasoning: w/o FECorr** (removing the LLM's factual error correction), as well as replacing our LLM reasoning approach with **CoT-SC** and **ReAct**.
- **Reasoning Graph Retrieval: w/o Pruning**, as well as replacing random-walking with **CART** [57], and **PageRank** [70], where CART and PageRank are representative graph pruning algorithms.
- **Text Matching Similarity:** Replacing TF-IDF with **Levenshtein Distance** [22], **Euclidean distance**, and **Cosine Distance**, where Euclidean and Cosine distance utilize the Word2Vec [58] to embed the two sentences s_x and s_y with the mean value of word embeddings, then calculate the similarity between the two sentence embeddings (i.e., $sim(w2v(s_x), w2v(s_y))$).

where the symbol **w/o** means removing the component from the VulRTex, and the variant without **w/o** means replacing the component with the corresponding variants.

The results of the ablation study show that: From Fig. 6 (a) and (c), we can see that the variants in **rich-text information** (-16.7% F1, -12.2% Macro-F1) and **reasoning graph retrieval** (-16.5% F1, -10.7% Macro-F1) lead to a large decrease in vulnerability-related IR Identification and CWE-ID prediction. Among the variants, removing the page screenshot analysis **w/o *[SCR]*** (-17.6% F1, -13.0% Macro-F1) and removing the random-walking pruning (-19.2% F1, -12.9% Macro-F1) has the largest decrease. From Fig. 6 (b) and (d), we can see that, compared with the other two types of variants in (a) and (c), the variants in **LLM reasoning approaches** (-14.3% F1, -9.6% Macro-F1) and **text matching similarity** (-12.1% F1, -6.1% Macro-F1) lead to a moderate decrease in vulnerability-related IR identification and CWE-ID prediction. Among the variants, replacing TF-IDF similarity with Levenshtein distance (-14.6% F1, -8.6% Macro-F1) has the largest decrease.

Comparison on OCR Tools. To analyze the elements in the screenshots, we have compared the performances of OCR tools and MLLMs as follows:

- **OCR Tools:** We have compared Baidu-OCR, IFlyTek-OCR, Tesseract-OCR, and Tencent-OCR. All four OCR tools a practically useful during our previous research and projects for analyzing the semantic information in the images.

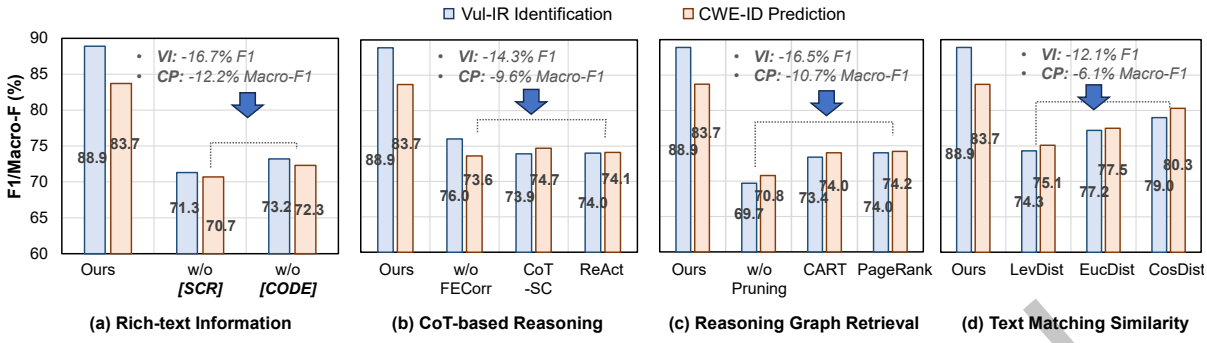


Fig. 6. The contribution of four types of variants to the VulRTex's performances.

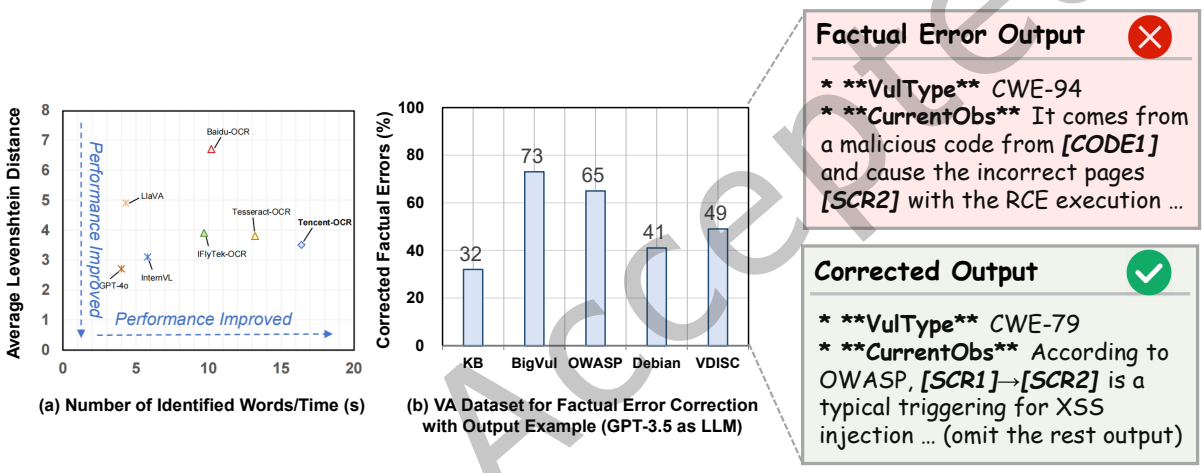


Fig. 7. The selection of OCR tools, as well as the contribution of VA datasets to factual error correction.

- **MLLM Tools:** We have compared the LLaVA, GPT-4o, and InternVL. The usage of these four MLLMs is different from the baselines, because their duty is only describing the information in the rich-text information, e.g., the elements in the screenshots, with plain text.

To evaluate the efficiency of OCR and MLLM Tools, we have used two metrics: **(1) Number of Identified Words/Time**, which measures the time cost of these tools; and **(2) Average Levenshtein Distance**, which measures the accuracy of parsed plain-text information by calculating the similarity with the manually-labeled descriptions. Fig. 7 (a) shows the performances of the tools. We can see that Tencent OCR API has high accuracy (rank 3rd) and time efficiency (rank 1st) in the rich-text parsing, which can accurately analyze more page elements in unit time. Therefore, we have chosen the Tencent OCR in the VulRTex's experiments.

Comparison on Factual Error Correction. In addition to variants and OCR tools, we analyze the contribution of VA datasets to factual error correction. We first analyzed the samples of factual errors that VulRTex successfully removed in reasoning, then we analyzed the proportion of VA datasets from which the samples came and calculated the proportion. We utilize the prompt $P_{correct}$ in Section 3.1.3 to correct the errors.

Table 6. The details of vulnerability-related IRs that are newly identified by VulRTex and assigned to CVE-IDs.

OSS Project	Stars	#IR	Statistics of Assigned CVE-IDs			Example of Assigned CVE-IDs				
			#Identified	#CVE-Assigned	#Potential	Vul-IR	IR's Type	CWE-ID	CVE-Assigned	Identified Date
Rimedo-ts [19]	22	1	1	1/1 (100.0%)	0/1 (0.0%)	#16	Plain-Text	CWE-119	CVE-2024-34049	02/20/2024 (**-69 Days**)
Carla [20]	11.3K	300	6	1/6 (16.7%)	2/6 (33.3%)	#7025	Rich-Text	CWE-119	CVE-2024-33903	01/10/2024 (**-110 Days**)
Hyprland [15]	21.1K	720	2	1/2 (50.0%)	1/2 (50.0%)	#5787	Plain-Text	CWE-362	CVE-2024-33904	04/18/2024 (**-11 Days**)
React [10]	229K	197	4	0/4 (0.0%)	1/4 (25.0%)	#31174	Rich-Text	CWE-1333	-	10/10/2024
Python-jose [18]	1.5K	10	3	2/3 (66.7%)	0/3 (0.0%)	#344	Rich-Text	CWE-400	CVE-2024-33664	03/13/2024 (**-43 Days**)
Xxl-job [21]	27.5K	130	6	2/6 (33.3%)	2/6 (33.3%)	#3375	Rich-Text	CWE-918	CVE-2024-24113	01/14/2024 (**-25 Days**)
Jerryscript [16]	6.9K	12	2	1/2 (50.0%)	1/2 (50.0%)	#5135	Rich-Text	CWE-671	CVE-2024-33255	03/29/2024 (**-28 Days**)
Node-server [14]	375	16	1	1/1 (100.0%)	0/1 (0.0%)	#159	Rich-Text	CWE-20	CVE-2024-32652	04/18/2024 (**-1 Day**)
Hugo [13]	75.5K	115	2	1/2 (50.0%)	1/2 (50.0%)	#12396	Rich-Text	CWE-20	CVE-2024-32875	04/20/2024 (**-3 Days**)
Kubernetes [17]	111K	597	3	1/3 (33.3%)	1/3 (33.3%)	#124336	Rich-Text	CWE-285	CVE-2024-3177	04/20/2024 (**-2 Days**)
Total		2,098	30	11/30 (36.7%)	9/30 (30.0%)	-	-	-	-	-

Fig. 7 (b) shows the results of the VA datasets to correct the GPT-3.5's output factual errors. We can see that all the VA dataset's items can correct the output factual errors, where BigVul has the best performance, which corrects 73% of factual errors. Moreover, we use an example to intuitively show the corrected results, which also uses the Fig. 1's example. In this case, GPT-3.5 is misled by the keyword "*RCE Execution*" and identifies the vulnerability as CWE-94 (Code Injection Vulnerability). CWE-94 is different from CWE-79 (XSS), because it may not require the adversaries to inject malicious script into the web pages. Even if we have added the rich-text information into the GPT-3.5, it still does not capture the necessary features, i.e., web page and `JavaScript` scripts. After we introduce the OWASP's golden knowledge, GPT-3.5 finds itself making a mistake and ignores these features, so it corrects the previous error outputs and identifies the CWE-79 vulnerability. This case intuitively shows the advantages and necessities of the VA dataset in the VulRTex.

Answering to RQ2

VulRTex outperforms all the variants in the ablation study. The variants in rich-text information and reasoning graph retrieval have the largest contributions, and the CoT-based reasoning and text-matching similarities have moderate contributions on average. Besides, the selection of OCR tools will affect the parsing efficiency of rich-text information, where we select the best-performing Tencent OCR API in the experiments. Finally, we find that the VA datasets can contribute to correcting the factual errors in the VulRTex's outputs, which can improve LLM's understanding of different vulnerabilities' triggering processes.

5.3 RQ3: Performances on Identifying Emerging Vulnerabilities in Open-source Projects

To analyze the performances of VulRTex on identifying emerging vulnerabilities, we deploy VulRTex and continuously apply it to track the GitHub IRs within these projects proposed after *Jan 1st, 2024*. Since all the IRs in the previous experiments were collected before this date, these newly tracked GitHub IRs are **not included** in the collected dataset in Section 4.1. Then, we track the IRs with the issue-tracking system (e.g., Bugzilla [4], etc.) on these projects, and observe 30 IRs that may potentially describe vulnerabilities with VulRTex. Then, we notify the OSS project's owners and IR authors through their emails or submit target IR comments if emails are not reserved. We ask the IR authors and project owners the following two questions to verify whether our identified IRs really contain the vulnerability, and require assigning CVE-IDs for these vulnerabilities:

- ***Q1:*** *Can you manually identify whether the given IR we provide really contains the vulnerability with the corresponding CWE-ID? Please answer [Yes/No].*
- ***Q2:*** *If this IR contains the vulnerability, can you report it to the CVE for vulnerability disclosure?*

We pay attention to the popular OSS projects on GitHub with multiple participants and widely used in 2024 based on the *Gitstar Ranking* [12], and remove some personal projects with fewer star ratings. The column "**Statistics of Assigned CVE-IDs**" of Table 6 shows the number of identified IRs and the CVE-Assigned IRs.

The newly identified vulnerability-related IRs belong to the 10 unique projects, from ***Rimedo-ts*** to ***Kubernetes***. Among them, most of the projects obtain over 1K stars, and some projects (e.g., ***React***, ***Hugo***, and ***Kubernetes***) are among the top 100 large-scale projects with a large user base. We can see that 30 of 2,098 IRs are identified as vulnerability-related IRs. Among these identified IRs, 11 of them (36.7%) are assigned CVE-IDs after vulnerability identification, and 9 of them (30.0%) are **potentially-assigned CVE**, which means that project owners admit that these IRs are vulnerability-related based on the feedback from the questionnaire, but have not been assigned CVE-IDs yet. These results indicate that VulRTex can identify emerging vulnerabilities from target IRs.

The column "**Statistics of Assigned CVE-IDs**" of Table 6 shows the details of CVE-Assigned IRs. We can see that, the date of vulnerability identification is earlier than the CVE-Assigned date. For some open-source projects (i.e., ***Rimedo-ts***, ***Carla***, and ***Python-jose***), the identification time can even be 40 days ahead of the CVE-Assigned time. It is worth mentioning that VulRTex can identify only a few vulnerability-related IRs in large-scale projects, mainly because these projects are well-maintained and the proportion of these IRs is small. Meanwhile, most vulnerability-related IRs we identified were disclosed in April because CVEs usually disclose the vulnerabilities within a certain period. In these cases, our method can successfully identify these vulnerabilities, which further illustrates the ability of VulRTex to identify emerging vulnerabilities in practice.

Answering to RQ3

VulRTex can identify the emerging vulnerabilities from large-scale OSS projects. Among these 30 newly identified vulnerability-related IRs from 10 representative projects, 11 IRs (36.7%) are assigned CVE-IDs, and 9 IRs (30.0%) will potentially be assigned CVE-IDs from the feedback from project owners.

6 Discussion

6.1 Analysis of Hyper-Parameters

In the VulRTex's overall framework, two different hyperparameters determine its performance on the vulnerability identification tasks, i.e., ***Historical IR's Proportion*** and ***TF-IDF Similarity Threshold*** θ_{sim}. To analyze the sensitivity of these two parameters, we conduct the overall evaluation, cross-project evaluation, and cross-CWE-ID evaluation of the model performances.

It is worth noticing that parameter θ_{out} aims to evaluate the performance of VulRTex on the dataset when the positive and negative samples are imbalanced, which reflects the result of the AUPRC and AUROC metrics (i.e., area under the PR-Curve with different θ_{out} threshold), so we do not reanalyze its contribution in this section.

Overall Sensitivity Analysis. For the proportion of historical IRs, we choose the proportion from 30% to 90%, with intervals of 10%. For the θ_{sim}, we choose the value from 0.50 to 0.90, with the intervals 0.05.

- **Overall Contribution of Historical IR's Proportion:** Fig. 8 (a) shows the effect of hyperparameters of VulRTex on average. We can see that historical IR's proportion affects the model performance. From 30% to 60%, the performances of VulRTex have a large increase with +6.4% F1 and +6.9% Macro-F1; after 60%, the fluctuations are small, with less than ±0.1% F1/Macro-F1.
- **Overall Contribution of TF-IDF Similarity Threshold:** Fig. 8 (b) shows the effect of θ_{sim} on average. We can see that the similarity threshold affects the model performance. VulRTex achieves the optimal performance when $\theta_{sim} = 0.70$, which is higher than other values with over +1.0% F1/Macro-F1 performances on average.

In summary, we finally chose the proportion 60% and $\theta_{sim} = 0.70$ in the VulRTex's experiments, which is sufficient to achieve optimal performance in the vulnerability identification.

Sensitivity Analysis across Projects and CWE-IDs. Since the projects and CWE-IDs are sparsely distributed in the testing dataset of target IRs with the long-tail distribution, we have chosen the Top-3 projects and CWE-IDs with the largest number of vulnerability-related IRs.

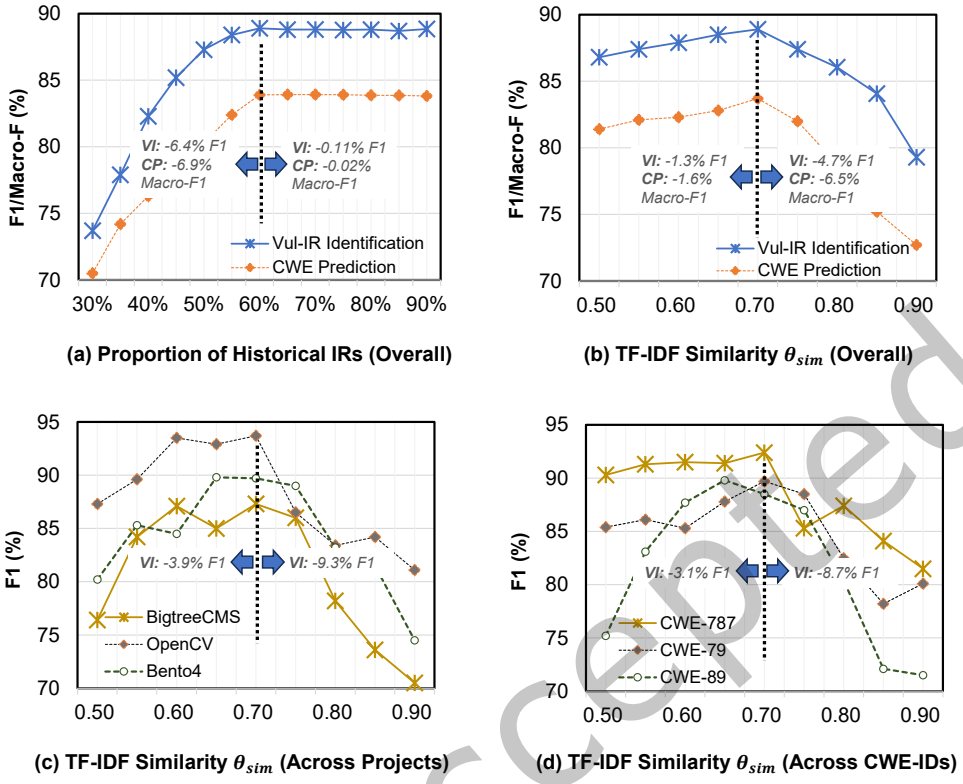


Fig. 8. The sensitivity analysis of hyperparameters to VulRTex's performances.

We have only evaluated the results of vulnerability identification because the CWE-IDs of some projects are very sparsely distributed, resulting in randomness of the final weighted Macro-F1. We also choose the θ_{sim} values from 0.50 to 0.90 with intervals of 0.05, and the results are shown as follows:

- **Contribution of TF-IDF Similarity across Projects:** We have chosen *BigtreeCMS* (118 Vul-IRs), *OpenCV* (80 Vul-IRs), and *Bento4* (219 VulIRs). All these projects account for 26% of the target IRs. We can see that 2/3 projects' F1 performances achieve an optimal value when $\theta_{sim} = 0.7$. From 0.50 to 0.7, the performances of VulRTex have an increase of +3.9% F1; after 0.7, on average, the F1 decreases by 9.3%. Even for the Bento4 project, the performance difference between $\theta_{sim} = 0.65/0.70$ is less than 3.0% F1.
- **Contribution of TF-IDF Similarity across Projects:** We have chosen *CWE-787* (156 Vul-IRs), *CWE-79* (220 Vul-IRs), and *CWE-89* (307 VulIRs). All these CWE-IDs account for 43% of the target IRs. We can see that We can see that 2/3 CWE-IDs' F1 performances achieve an optimal value when $\theta_{sim} = 0.7$. From 0.50 to 0.7, the performances of VulRTex have an increase of +3.1% F1; after 0.7, on average, the F1 decreases by 8.7%. The counterexample of CWE-89 may be due to the diverse way of illustrating the process of *SQL injection*, and we will analyze it in the future.

In summary, we believe that choosing $\theta_{sim} = 0.70$ for VulRTex can help it achieve optimal values for most projects and CWE-IDs in the experiments. The above results also provide a basis for our parameter setting.

6.2 Application of VulRTex on Other Languages' IRs

In this section, we have analyzed the ability of VulRTex on Chinese, Japanese, and French IRs. We find that GHArchive's IRs may not be applicable, which comes from GitHub and CVE-IDs. Only a few IRs (in our manual analysis, we find that fewer than 10 IRs) contain words, phrases, or sentences in Chinese, Japanese, and French, but the other parts of the main body still use English to describe the vulnerability triggering process.

To evaluate VulRTex's performance across languages, we have made the following extension.

- **Database Extension:** We have introduced other vulnerability databases, such as evaluated CNNVD (Chinese), JPCERT/CC (Japanese), and ANSSI (French). These databases are maintained by these countries' security departments, which enhances the language diversity.
- **OSS Community Extension:** We have introduced other OSS communities, such as Gitee and GitLab etc. The number of IRs increases with these new communities, and the formats of these new IRs are similar to the original GitHub IRs, so VulRTex is applicable to identify the vulnerabilities from them.

In total, we collect 22 Chinese, 19 Japanese, and 14 French vulnerability-related IRs from these new databases and communities, where 10, 13, and 11 of them contain rich-text information. Since the number of vulnerabilities is small, we reuse the vulnerability reasoning database prepared with GHArchive, and apply VulRTex to identify the vulnerabilities.

Table 7. The performance of GPT-3.5+VulRTex on IRs on three other languages, i.e., Chinese, Japanese, and French.

Language	Plain-Text IRs			Rich-Text IRs		
	#Vul-IR	#VI (Ratio%)	#CP (Ratio%)	#Vul-IR	#VI (Ratio%)	#CP (Ratio%)
Chinese	12	8/12 (66.7%)	7/12 (58.3%)	10	8/10 (80.0%)	8/10 (80.0%)
Japanese	6	6/6 (100.0%)	6/6 (100.0%)	13	9/13 (69.2%)	9/13 (69.2%)
French	3	3/3 (100.0%)	3/3 (100.0%)	11	10/11 (90.9%)	7/11 (63.6%)
Total	*21*	*17/21 (90.0%)*	*16/21 (76.2%)*	*34*	*27/34 (79.4%)*	*24/34 (70.6%)*

Table 7 shows the experimental results of GPT-3.5+VulRTex on three new languages' IRs, where **#VI** and **#CP** mean the number of accurately identified vulnerabilities and predicted CWE-IDs. We can see that, for plain-text and rich-text IRs, VulRTex can identify over 79% vulnerability-related IRs, and predict CWE-IDs with over 70%. For each language, the performances of CWE-ID prediction results on rich-text Japanese and French IRs are lower than Chinese IRs, which is because the pre-trained data of the Tencent OCR tool is mostly in English and Chinese, so it may not be accurate enough for French and Japanese screenshots. Finally, we find that these results are lower than the GHArchive (Table 4), which may be due to we only reuse the vulnerability reasoning dataset without adding historical IRs. We will enlarge the dataset with other languages' IRs.

6.3 Advantages of VulRTex

The benefits of VulRTex come from three aspects: the ability to analyze rich-text information, the correction of factual errors, and the utilization of RAG to improve efficiency. With these three advantages, VulRTex can not only accurately identify vulnerabilities in rich-text IRs but also outperform the baselines on plain-text IRs.

- **Advantage-1: Analysis of Rich-Text Information.** Existing approaches, such as MemVul, cannot identify vulnerabilities in few-text IRs, and they need to track the source code in the repositories for further analysis. In comparison, VulRTex indicates that there will be rich-text elements, such as page screenshots and code snippets that can be utilized to complement the missing information in the few-text IRs and help identify the vulnerability triggering paths.

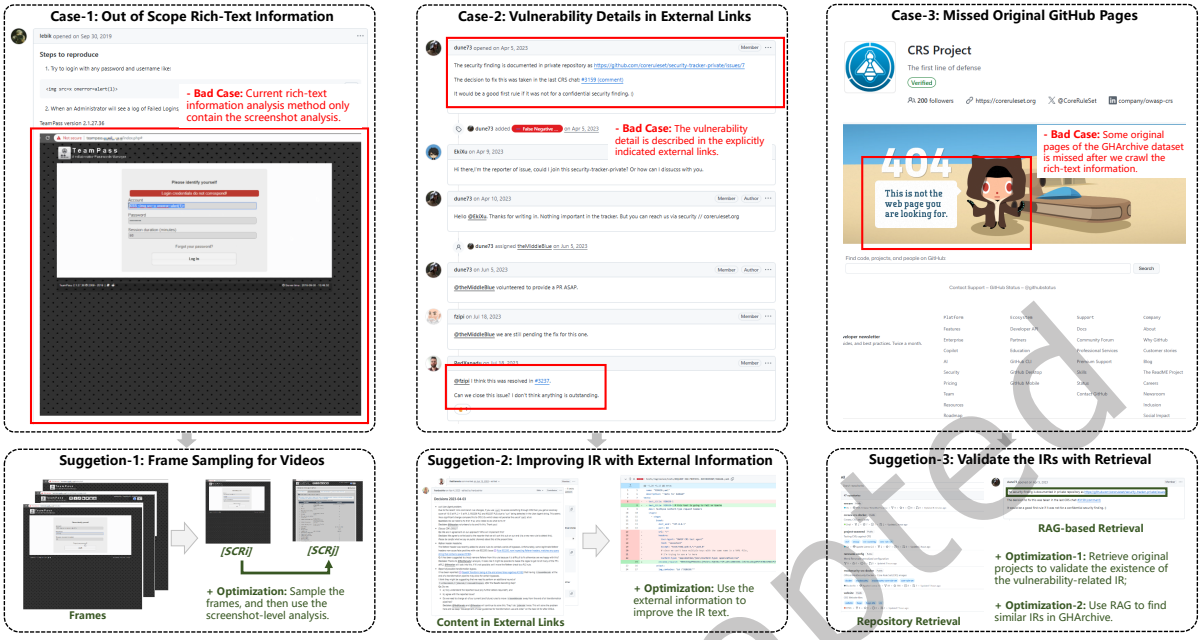


Fig. 9. The example of VulRTex's incorrect prediction.

- **Advantage-2: Correction of Factual Errors.** Since the IR may incorrectly describe how the vulnerability is triggered, and LLM itself may be trained on outdated and noisy datasets, the output may have some factual errors. With the help of factual error correction, the reasoning graphs of rich/plain-text IRs will be in line with the real-world situation, thus improving the performance of vulnerability identification.
- **Advantage-3: Utilization of RAG's Thought.** Based on the reasoning graph retrieval, the LLMs will understand how the previous developers and users describe the triggering logic of similar vulnerabilities in their IRs, and the LLMs will understand semantic information in the relevant texts and rich-text elements. Therefore, the thought of RAG not only improves the efficiency of VulRTex, but also enables LLM to understand the details of vulnerabilities and enhance the quality of its generated text in the security field.
- **Advantage-4: Lower Computational Requirements.** We have utilized the locally deployed models (i.e., LLaMA and Qwen2VL) and models' API (i.e., GPT-3, GPT-3.5, GPT-4o, and DeepSeek-R1). The GPU cost of locally deployed models with VulRTex is 67.3% on average (1×A6000), which is more than the original model (44.5% on average), but much lower than the average cost of reasoning baselines (CoT-SC, ReAct, and DS-Agent, 86.7% on average). For the memory cost, VulRTex is 55.7% on average for all the APIs, higher than the original baselines (54.4% on average) and lower than the reasoning baselines (69.3% on average). In summary, we believe that VulRTex has a lower computational requirement compared with baselines.

6.4 Incorrect Prediction of VulRTex

Although VulRTex can accurately identify the vulnerabilities from rich-text IRs, there are still 15.0% of IRs that are incorrectly predicted in our experiment. As is illustrated in Fig. 9, we manually inspect these incorrect predictions and find the following three reasons that limit the performance of VulRTex:

- Case-1: Out-of-Scope Rich-Text Information (4.5%): The type of rich-text information is not included in our approach (note that, in our dataset, all the bad cases contain videos). For example, the IR *TeamPass/issues/2688* [64] utilizes the video stream to present the steps of XSS injection in the TeamPass project, but VulRTex cannot analyze the video stream, which leads to the incorrect prediction of CWE-ID.

+ Potential Solution to Case-1: To enable VulRTex to analyze the out-of-scope rich-text information, we can use the frame sampling method to transfer the original videos to a sequence of screenshots. Then, we select the key frames in the video and analyze their transitions, which reflect the triggering process of the vulnerabilities.

- Case-2: Vulnerability Details in External Links (6.5%). Some original pages of vulnerability-related IRs contain external links to describe the details of how the vulnerability is triggered. For example, in the issue *coreruleset/issues/3191* [7], the original vulnerability details are involved in other comment links. VulRTex cannot retrieve this knowledge from the external links, which leads to the incorrect prediction.

+ Potential Solution to Case-2: We can use the external information to complement the missing information and improve the original IR. For example, the `Python-Selenium` library provides the ability to open an external link and crawl the data on this page. Sometimes, the IR needs to open multiple external links, such as the example in Fig. 9, where LLM can identify the types with the code in commit messages.

- Case-3: Missed Original GitHub Pages (4.0%). Some original pages of vulnerability-related IRs are deleted or hidden by authors, which leads to incorrect predictions. For example, the page of coreruleset's issue *security-tracker-private/issues/7* [9] has been set as private, so the page has "*404 Error*". This bad case is different from Case-2, because we cannot obtain the vulnerability details through the external links directly in this case.

+ Potential Solution to Case-3: The reason why we cannot obtain the original pages may be the following two reasons: the repository is set to private or has been deleted. To validate whether we need to further analyze this issue or discard it, we can introduce the RAG thought, which retrieves the original repositories and similar IRs from the author's GitHub page. If the project is deleted, we may consider the IR is outdated and discard it.

6.5 Threats to Validity

The threats of VulRTex may come from three aspects: Internal, External, and Constructive threats. We introduce these three types of threats and their potential mitigations as follows.

Internal Threats. The first internal threat comes from dataset preparation. We only refer to the CVE to analyze whether this vulnerability is disclosed, but some vulnerabilities may only be disclosed in other security databases, such as CAPEC [29]. To alleviate this threat, we manually inspect 100 security GitHub IRs that are disclosed on CAPEC and find that 2/100 are not disclosed by CVE and are incorporated into our dataset simultaneously. Another threat comes from the TF-IDF that may introduce noisy data in correcting LLM factual errors. The LLM itself can remove some noisy data with its reasoning ability, which alleviates this threat. We have randomly sampled 100 vulnerability-related IRs from our dataset with target IRs, and manually inspected whether VulRTex can retrieve the matched golden knowledge in the knowledge $Know_t$. We find that 21/100 IRs may have LLM's factual errors, and all the IRs (21/21) are corrected with $Know_t$. Therefore, the impact of internal threats is small.

External Threat. The external threat comes from the potentially incorrect labels in the dataset. For example, CVE-2018-17566 indicates that *top-think/think/issues/858* is the vulnerability-related IR with the "SQL injection" vulnerability, but in the following comments, other developers indicate that this vulnerability may not exist in practice. The impact of this threat is small due to the small proportion of samples (60/4,003), and we plan to report them to the CVE for further validation.

Constructive Threat. The constructive threat mainly comes from the metrics. We choose precision, recall, and F1-Score to evaluate the vulnerability-related IR identification, and choose Macro-P, Macro-R, and Macro-F1 to evaluate the performance on CWE-ID prediction. We manually restore the rich-text information by retrieving the

original IR from GitHub while calculating these metrics. This threat is mitigated by the fact that all the GitHub IRs are reviewed and discussed by our team members in the dataset preparation.

7 Related Works

In this section, we will introduce the related works in three aspects. First, we introduce the automatic vulnerability detection methods before the emergence of LLMs, which mainly used rules of taint analysis and DL methods to locate the vulnerability code lines in the programs. Then, we discuss the evolution of LLM agents, which interact with the external environments to resolve complex queries and help build embodied intelligence systems. Moreover, we introduce the application of LLM agents on vulnerability detection and repairing in recent works.

7.1 Automatic Vulnerability Identification

The automatic detection and identification of vulnerabilities has been investigated by researchers. Automatic vulnerability detection aims to determine whether there are malicious codes that contain vulnerabilities during the project development [26, 35, 43, 46, 52, 56, 113]. The researchers first proposed the statistic, dynamic, and hybrid techniques to detect the vulnerabilities with rules [32, 42, 51]. To improve detection accuracy and reduce the cost of manually designing rules, researchers have introduced machine learning (ML) approaches to detect the vulnerabilities, which combine the features extracted from codes and feed them into the basic ML models to predict the vulnerability types [82, 86, 102]. With the development of DL and LLM, researchers have introduced these novel models to automatically build code features and improve the efficiency of vulnerability detection tools [50, 50, 53, 92]. Automatic vulnerability identification helps security practitioners identify whether artifacts (e.g., GitHub IRs, bug reports, etc.) submitted by developers actually contain vulnerabilities. Some researchers utilized text-mining methods to explore the security bug reports to identify the vulnerabilities [34, 97, 98, 101], while other works focused on reducing the negative impact of vulnerability identification from class unbalancing [69, 72, 75, 87]. Our work focuses on identifying vulnerabilities in IRs.

Different from these previous works, the current vulnerability detection tasks encounter a complex description of the vulnerability triggering process, such as utilizing rich-text information and natural language descriptions to illustrate how the vulnerabilities occur in their daily OSS development. Therefore, we introduce LLM's reasoning ability to analyze the meaningful information involved in rich-text information, and combine the reasoning graph retrieval method to improve the accuracy of identifying vulnerability-related IRs and predict the CWE-IDs.

7.2 LLM Agent and Embodied Intelligence System

The LLM and agent systems have been developed in recent years, which facilitate the development of software engineering. Researchers have investigated the application of LLMs' reasoning and agents on different research tasks, where some of these works were based on textual description, and they utilized LLM agents to analyze the text-based reasoning logic. Wang et al. [93] analyzed the logic of dialogues and applied LLM agents in the communications. Nan et al. [63] integrated the reasoning and action in LLM agents to describe the database question answering. Since the LLM agent systems could interact with the external tools and databases, the researchers extended the application scope of them, and helped build the embodied intelligence systems. These advanced systems combined visual information and proposed the embodied LLM agents to analyze more complex visual-related tasks. Zheng et al. [110] proposed Steve-Eye, an embodied agent that analyzes visual perception in open worlds. Cherakara et al. [28] proposed the FurChat, which is a conversational-based embodied agent that combines open and closed domain dialogues with facial expressions. Schumann et al. [83] proposed the VELMA, which is an embodied agent that analyzes the language navigation and vision in street view. Ma et al. [55] proposed the LASER, which utilizes the LLM agent to analyze the website navigation.

However, it is unclear whether the application of the LLM agent systems can be applied to vulnerability detection tasks. The current LLM agents are difficult to analyze the process of vulnerability triggering and output hallucinations, such as factual errors that do not comply with the common sense of vulnerability detection. In comparison, VulRTex incorporates the reasoning graph retrieval from the thought of RAG to improve the accuracy of LLM agents in analyzing rich-text information.

7.3 LLM Agent System for Vulnerability Identification

In recent years, researchers have applied LLM agents to vulnerability detection and repair. Bouzenia et al. [25] propose the RepairAgent, which was the first work that repaired vulnerable programs with LLM-based autonomous agents and a finite state machine, which fixed 164 vulnerabilities (including 39 new vulnerabilities) on the Defects4J dataset; Huang et al. [40] proposed the LLM agents with a two-step counterfactual suggestion, where the coach module guides the LLM to explore multiple attack paths. The vulnerability coverage of this method increased by 31%, and the repair strategy effectiveness increased by 32%. Tihanyi et al. [90] combined Bound Model Checking with LLM agents, which used bounded models to generate counterexamples and locate the vulnerabilities. This work achieves 80% fixing rate of "buffer overflow" and "pointer dereference failure" vulnerabilities on over 1000 C code examples. However, these works face limitations in practical usage. Wang et al. [94] proposed the VulnRepairEval framework, which was a rigorous benchmark for evaluating the repair capabilities of LLM agents, moving beyond simple functional tests to exploit-based validation. Based on the evaluation of 12 mainstream LLMs, the optimal model repair rate is only 21.7%, revealing the overestimation of the current models. To address this issue, researchers proposed the Multi-Agent System (MAS), where different agents specialize in tasks like code auditing, static analysis, or fuzzing, collaborating to find vulnerabilities more efficiently than a single agent. Zhao et al. [109] proposed MADE (Malicious Agent Detection), which detected and removed malicious agents attempting to degrade the system through adversarial attacks, improving the safety in collaborative perception scenarios (e.g., autonomous driving).

Until now, we find that the number of related works on applying LLM agents to vulnerability detection is small, and the technical details of them rely on the project's source code. If some vulnerabilities are reported without open-sourcing their commit messages, it will be difficult to analyze the vulnerability types and develop the repair methods. In comparison, our work considers both time efficiency and performance of vulnerability identification. VulRTex is an agent system that can learn from the historical IRs with RW-based pruning and remove factual errors, assisting security practitioners in discovering and fixing vulnerabilities on time.

8 Conclusion and Future Work

In this paper, we propose the VulRTex to identify vulnerability-related IRs with rich-text information. VulRTex first utilizes the reasoning ability of LLMs to prepare the Vulnerability Reasoning Database from historical IRs. Then, VulRTex retrieves the most relevant reasoning graphs from the prepared reasoning database to guide LLM in identifying vulnerabilities from target IRs. Experiments conducted on 973,572 IRs show that VulRTex achieves the highest performance when the dataset is imbalanced, outperforming the best baseline with +11.0% F1 and +20.2% AUPRC, with 2x lower time cost than the baseline reasoning approaches. VulRTex also has the highest performance on CWE-ID prediction, outperforming the best baseline with +10.5% Macro-F1. Furthermore, VulRTex has been applied to identify the 30 emerging vulnerability-related IRs across 10 projects, and 11 of them are finally assigned CVE-IDs.

In the future, we plan to identify IRs from other collaborative platforms, such as GitLab, Gitee, etc., and introduce insecure code commits & patches that relate to the vulnerabilities to continuously improve the capability of VulRTex in more open-source projects. With the development of post-training, we also plan to design vulnerability-oriented rewards to optimize the model, which alignment it with the real-world feedback.

Acknowledgments

We sincerely appreciate the reviewers for their insightful suggestions. This work was supported by the National Natural Science Foundation of China (Key Program) No.62232016, National Natural Science Foundation of China Grant No.62332001, No.62272445, No.62402484, Youth Innovation Promotion Association Chinese Academy of Sciences, Basic Research Program of ISCAS Grant No. ISCAS-JCZD-202405, and partially supported by the University of Queensland NSRSG Grant NS-2201 and Oracle Labs.

References

[1] 2018. ISO/IEC 29147:2018: Security techniques - Vulnerability disclosure. https://www.iso.org/standard/72311.html.
[2] 2019. XSS and CSRF in Blocks. https://github.com/daylightstudio/fuel-cms/issues/536.
[3] 2020. XSS in cmd.php for 1.2.5. https://github.com/leenooks/phpldapadmin/issues/130.
[4] 2023. Bugzilla. https://www.bugzilla.org/.
[5] 2023. Common vulnerabilities and exposures. https://cve.mitre.org/.
[6] 2023. Common weakness enumeration. https://cwe.mitre.org/.
[7] 2023. Fix C9K-230327. https://github.com/coreruleset/coreruleset/issues/3191.
[8] 2023. GHArchive. https://www.gharchive.org/.
[9] 2023. Page Not Found - GitHub. https://github.com/coreruleset/security-tracker-private/issues/7.
[10] 2024. Facebook/React. https://github.com/facebook/react.
[11] 2024. Gitstar Ranking. https://www.akto.io/cves/vendor/debian.
[12] 2024. Gitstar Ranking. https://gitstar-ranking.com/repositories.
[13] 2024. Gohugoio/Hugo. https://github.com/gohugoio/hugo.
[14] 2024. Honojs/Node-server. https://github.com/honojs/node-server.
[15] 2024. Hyprwm/Hyprland. https://github.com/hyprwm/Hyprland.
[16] 2024. Jerryscript-project/Jerryscript. https://github.com/jerryscript-project/jerryscript.
[17] 2024. Kubernetes/Kubernetes. https://github.com/kubernetes/kubernetes.
[18] 2024. Mpdavis/Python-jose. https://github.com/mpdavis/python-jose.
[19] 2024. Onosproject/Rimedo-ts. https://github.com/onosproject/rimedo-ts.
[20] 2024. Onosproject/Rimedo-ts. https://github.com/carla-simulator/carla.
[21] 2024. Xuxueli/Xxl-job. https://github.com/xuxueli/xxl-job.
[22] Zeeshan Afzal, Johan Garcia, Stefan Lindskog, and Anna Brunström. 2018. Slice Distance: An Insert-Only Levenshtein Distance with a Focus on Security Applications. In *9th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2018*. IEEE, 1–5.
[23] Atlassian. 2023. Jira, Issue & Project Tracking Software. https://www.atlassian.com/software/jira.
[24] Leyla Bilge and Tudor Dumitras. 2012. Before we knew it: an empirical study of zero-day attacks in the real world. In *the ACM Conference on Computer and Communications Security, CCS'12*. ACM, 833–844.
[25] Islem Bouzenia, Premkumar T. Devanbu, and Michael Pradel. 2025. RepairAgent: An Autonomous, LLM-Based Agent for Program Repair. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*. IEEE, 2188–2200. doi:10.1109/ICSE55347.2025.00157
[26] Saikat Chakraborty, Rahul Krishna, Yangruibo Ding, and Baishakhi Ray. 2022. Deep Learning Based Vulnerability Detection: Are We There Yet? *IEEE Trans. Software Eng.* 48, 9 (2022), 3280–3296.
[27] Yizheng Chen, Zhoujie Ding, Lamya Alowain, Xinyun Chen, and David A. Wagner. 2023. DiverseVul: A New Vulnerable Source Code Dataset for Deep Learning Based Vulnerability Detection. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses, RAID 2023*. ACM, 654–668.
[28] Neeraj Cherakara, Finny Varghese, Sheena Shabana, Nivan Nelson, Abhiram Karukayil, Rohith Kulothungan, Mohammed Afil Farhan, Birthe Nesset, Meriam Moujahid, Tanvi Dinkar, Verena Rieser, and Oliver Lemon. 2023. FurChat: An Embodied Conversational Agent using LLMs, Combining Open and Closed-Domain Dialogue with Facial Expressions. In *Proceedings of the 24th Meeting of the Special Interest Group on Discourse and Dialogue, SIGDIAL 2023*. Association for Computational Linguistics, 588–592.
[29] T. M. Corporation. 2011. Common Attack Pattern Enumeration and Classification (CAPEC). http://capec.mitre.org/.
[30] Jesse Davis and Mark Goadrich. 2006. The relationship between Precision-Recall and ROC curves. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006) (ACM International Conference Proceeding Series, Vol. 148)*. ACM, 233–240.
[31] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei

Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, and S. S. Li. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *CoRR* abs/2501.12948 (2025). arXiv:2501.12948 doi:10.48550/ARXIV.2501.12948

[32] Dawson R. Engler, David Yu Chen, and Andy Chou. 2001. Bugs as Deviant Behavior: A General Approach to Inferring Errors in Systems Code. In *Proceedings of the 18th ACM Symposium on Operating System Principles, SOSP 2001*. ACM, 57–72.

[33] Jiahao Fan, Yi Li, Shaohua Wang, and Tien N. Nguyen. 2020. A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries. In *MSR '20*. ACM, 508–512.

[34] Michael Gegick, Pete Rotella, and Tao Xie. 2010. Identifying security bug reports via text mining: An industrial case study. In *Proceedings of the 7th International Working Conference on Mining Software Repositories, MSR 2010 (Co-located with ICSE)*. IEEE Computer Society, 11–20.

[35] Seyed Mohammad Ghaffarian and Hamid Reza Shahriari. 2017. Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. *Comput. Surveys* 50, 4 (2017), 1–36.

[36] Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. 2024. DS-Agent: Automated Data Science by Empowering Large Language Models with Case-Based Reasoning. *CoRR* abs/2402.17453 (2024). arXiv:2402.17453

[37] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. REALM: Retrieval-Augmented Language Model Pre-Training. *CoRR* abs/2002.08909 (2020). arXiv:2002.08909

[38] Nima Shiri Harzevili, Alvine Boaye Belle, Junjie Wang, Song Wang, Zhen Ming (Jack) Jiang, and Nachiappan Nagappan. 2025. A Systematic Literature Review on Automated Software Vulnerability Detection Using Machine Learning. *ACM Comput. Surv.* 57, 3 (2025), 55:1–55:36.

[39] Allen D Householder, Garret Wassermann, Art Manion, and Chris King. 2017. The cert guide to coordinated vulnerability disclosure. *Software Engineering Institute, Pittsburgh, PA* (2017).

[40] Junjie Huang and Quanyan Zhu. 2024. PenHeal: A Two-Stage LLM Framework for Automated Pentesting and Optimal Remediation. In *Proceedings of the Workshop on Autonomous Cybersecurity, AutonomousCyber 2024, Salt Lake City, UT, USA, October 14-18, 2024*, Ali Dehghantanha, Reza M. Parizi, and Gregory Epiphaniou (Eds.). ACM, 11–22. doi:10.1145/3689933.3690831

[41] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2023. A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions. *CoRR* abs/2311.05232 (2023).

[42] Jiyong Jang, Abeer Agrawal, and David Brumley. 2012. ReDeBug: Finding Unpatched Code Clones in Entire OS Distributions. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*. IEEE Computer Society, 48–62.

[43] Tiantian Ji, Yue Wu, Chang Wang, Xi Zhang, and Zhongru Wang. 2018. The coming era of alphahacking?: A survey of automatic software vulnerability detection, exploitation and patching techniques. In *2018 IEEE third international conference on data science in cyberspace (DSC)*. IEEE, 53–60.

[44] Ziyou Jiang, Lin Shi, Guowei Yang, and Qing Wang. 2023. SCPatcher: Mining Crowd Security Discussions to Enrich Secure Coding Practices. In *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023*. IEEE, 358–370.

[45] Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active Retrieval Augmented Generation. In *EMNLP'23*. Association for Computational Linguistics, 7969–7992.

[46] Gong Jie, Kuang Xiao-Hui, and Liu Qiang. 2016. Survey on software vulnerability analysis method based on machine learning. In *2016 IEEE first international conference on data science in cyberspace (DSC)*. IEEE, 642–647.

[47] Dongkyu Kim, Byoungwook Kim, Donggeon Han, and Matous Eibich. 2024. AutoRAG: Automated Framework for optimization of Retrieval Augmented Generation Pipeline. *CoRR* abs/2410.20878 (2024). arXiv:2410.20878

[48] Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *NeurIPS'20*.

[49] Zhen Li, Deqing Zou, Jing Tang, Zhihao Zhang, Mingqian Sun, and Hai Jin. 2019. A comparative study of deep learning-based vulnerability detection system. *IEEE Access* 7 (2019), 103184–103197.

[50] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. 2018. VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018*. The Internet Society.

[51] Hongliang Liang, Lei Wang, Dongyang Wu, and Jiuyun Xu. 2016. MLSA: A static bugs analysis tool based on LLVM IR. In *17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2016*. IEEE Computer Society, 407–412.

[52] Guanjun Lin, Sheng Wen, Qing-Long Han, Jun Zhang, and Yang Xiang. 2020. Software Vulnerability Detection Using Deep Neural Networks: A Survey. *Proc. IEEE* 108, 10 (2020), 1825–1848.
[53] Guanjun Lin, Jun Zhang, Wei Luo, Lei Pan, and Yang Xiang. 2017. POSTER: Vulnerability Discovery with Function Representation Learning from Unlabeled Projects. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, 2539–2541.
[54] Tao Liu and Longtao Zhang. 2018. Application of logistic regression in web vulnerability scanning. In *2018 International Conference on Sensor Networks and Signal Processing (SNSP)*. IEEE, 486–490.
[55] Kaixin Ma, Hongming Zhang, Hongwei Wang, Xiaoman Pan, and Dong Yu. 2023. LASER: LLM Agent with State-Space Exploration for Web Navigation. *CoRR* abs/2309.08172 (2023).
[56] Ruchika Malhotra. 2015. A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing* 27 (2015), 504–518.
[57] Richard G. Mathieu and Alan E. Turovlin. 2023. Lost in the middle - a pragmatic approach for ERP managers to prioritize known vulnerabilities by applying classification and regression trees (CART). *Inf. Comput. Secur.* 31, 5 (2023), 655–674.
[58] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *1st International Conference on Learning Representations, ICLR 2013*.
[59] MITRE. 2014. Adversarial Tactics, Techniques & Common Knowledge (ATT&CK). https://attack.mitre.org.
[60] MITRE. 2023. CWE-352: Cross-Site Request Forgery (CSRF) . https://cwe.mitre.org/data/definitions/352.html.
[61] MITRE. 2023. CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (4.13). https://cwe.mitre.org/data/definitions/79.html.
[62] MITRE. 2023. CWE-94: Improper Control of Generation of Code ('Code Injection'). https://cwe.mitre.org/data/definitions/79.html.
[63] Linyong Nan, Ellen Zhang, Weijin Zou, Yilun Zhao, Wenfei Zhou, and Arman Cohan. 2023. On Evaluating the Integration of Reasoning and Action in LLM Agents with Database Question Answering. *CoRR* abs/2311.09721 (2023).
[64] nilsteampassnet. 2019. Stored XSS in log of Failed Logins. https://github.com/nilsteampassnet/TeamPass/issues/2688.
[65] Marwan Omar and Stavros Shiaeles. 2023. VulDetect: A novel technique for detecting software vulnerabilities using Language Models. In *IEEE International Conference on Cyber Security and Resilience, CSR 2023*. IEEE, 105–110. doi:10.1109/CSR57506.2023.10224924
[66] OpenAI. 2023. Chatgpt: A language model for conversational AI. https://www.openai.com/research/chatgpt/.
[67] OpenAI. 2023. Vision - OpenAI. https://platform.openai.com/docs/guides/vision.
[68] OWASP. 2023. Open web application security project. https://www.owasp.org/index.php/MainPage.
[69] Tosin Daniel Oyetoyan and Patrick Morrison. 2021. An improved text classification modelling approach to identify security messages in heterogeneous projects. *Softw. Qual. J.* 29, 2 (2021), 509–553.
[70] Page, Lawrence, Brin, Sergey, and Terry. 1999. The PageRank citation ranking: Bringing order to the web. *stanford digital libraries working paper* (1999).
[71] Liuxuan Pan and Allan Tomlinson. 2016. A Systematic Review of Information Security Risk Assessment. *International Journal of Safety and Security Engineering* 6 (06 2016), 270–281.
[72] Shengyi Pan, Jiayuan Zhou, Filipe Roseiro Côgo, Xin Xia, Lingfeng Bao, Xing Hu, Shanping Li, and Ahmed E. Hassan. 2022. Automated unearthing of dangerous issue reports. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022*. ACM, 834–846.
[73] Jorge E. Pérez, Jessica Díaz, Javier García Martin, and Bernardo Tabuenca. 2020. Systematic Literature Reviews in Software Engineering - Enhancement of the Study Selection Process Using Cohen's Kappa Statistic. *J. Syst. Softw.* 168 (2020), 110657. doi:10.1016/j.jss.2020.110657
[74] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: online learning of social representations. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14*. ACM, 701–710.
[75] Fayola Peters, Thein Than Tun, Yijun Yu, and Bashar Nuseibeh. 2019. Text Filtering and Ranking for Security Bug Report Prediction. *IEEE Trans. Software Eng.* 45, 6 (2019), 615–631.
[76] Serena Elisa Ponta, Henrik Plate, Antonino Sabetta, Michele Bezzi, and Cédric Dangremont. 2019. A manually-curated dataset of fixes to vulnerabilities of open-source software. In *Proceedings of the 16th International Conference on Mining Software Repositories, MSR 2019*. IEEE / ACM, 383–387.
[77] Hongjin Qian, Peitian Zhang, Zheng Liu, Kelong Mao, and Zhicheng Dou. 2024. MemoRAG: Moving towards Next-Gen RAG Via Memory-Inspired Knowledge Discovery. *CoRR* abs/2409.05591 (2024). arXiv:2409.05591
[78] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, Bokai Xu, Zhen Zhang, Yining Ye, Bowen Li, Ziwei Tang, Jing Yi, Yuzhang Zhu, Zhenning Dai, Lan Yan, Xin Cong, Yaxi Lu, Weilin Zhao, Yuxiang Huang, Junxi Yan, Xu Han, Xian Sun, Dahai Li, Jason Phang, Cheng Yang, Tongshuang Wu, Heng Ji, Zhiyuan Liu, and Maosong Sun. 2023. Tool Learning with Foundation Models. *CoRR* abs/2304.08354 (2023). arXiv:2304.08354
[79] Bonan Ruan, Jiahao Liu, Weibo Zhao, and Zhenkai Liang. 2024. VulZoo: A Comprehensive Vulnerability Intelligence Dataset. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24)*. Association for Computing

Machinery, 2334–2337.
[80] Rebecca L. Russell, Louis Y. Kim, Lei H. Hamilton, Tomo Lazovich, Jacob Harer, Onur Ozdemir, Paul M. Ellingwood, and Marc W. McConley. 2018. Automated Vulnerability Detection in Source Code Using Deep Representation Learning. In *17th IEEE International Conference on Machine Learning and Applications, ICMLA 2018*. IEEE, 757–762.
[81] Gerard Salton and Christopher Buckley. 1988. Term-weighting Approaches in Automatic Text Retrieval. *Information Processing & Management* 24, 5 (1988), 513–523.
[82] Riccardo Scandariato, James Walden, Aram Hovsepyan, and Wouter Joosen. 2014. Predicting Vulnerable Software Components via Text Mining. *IEEE Trans. Software Eng.* 40, 10 (2014), 993–1006.
[83] Raphael Schumann, Wanrong Zhu, Weixi Feng, Tsu-Jui Fu, Stefan Riezler, and William Yang Wang. 2023. VELMA: Verbalization Embodiment of LLM Agents for Vision and Language Navigation in Street View. *CoRR* abs/2307.06082 (2023).
[84] Ensheng Shi, Yanlin Wang, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2021. CAST: Enhancing Code Summarization with Hierarchical Splitting and Reconstruction of Abstract Syntax Trees. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021*. Association for Computational Linguistics, 4053–4062.
[85] Lin Shi, Ziyou Jiang, Ye Yang, Xiao Chen, Yumin Zhang, Fangwen Mu, Hanzhi Jiang, and Qing Wang. 2021. ISPY: Automatic Issue-Solution Pair Extraction from Community Live Chats. In *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021*. IEEE, 142–154.
[86] Yonghee Shin, Andrew Meneely, Laurie A. Williams, and Jason A. Osborne. 2011. Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities. *IEEE Trans. Software Eng.* 37, 6 (2011), 772–787.
[87] Rui Shu, Tianpei Xia, Jianfeng Chen, Laurie A. Williams, and Tim Menzies. 2021. How to Better Distinguish Security Bug Reports (Using Dual Hyperparameter Optimization). *Empir. Softw. Eng.* 26, 3 (2021), 53.
[88] Andrea Stocco, Michael Weiss, Marco Calzana, and Paolo Tonella. 2020. Misbehaviour prediction for autonomous driving systems. In *ICSE '20: 42nd International Conference on Software Engineering*. ACM, 359–371.
[89] Tencent. 2023. OCR, Tencent Cloud. https://www.tencentcloud.com/document/product/1045/49147.
[90] Norbert Tihanyi, Yiannis Charalambous, Ridhi Jain, Mohamed Amine Ferrag, and Lucas C. Cordeiro. 2025. A New Era in Software Security: Towards Self-Healing Software via Large Language Models and Formal Verification. In *IEEE/ACM International Conference on Automation of Software Test, AST@ICSE 2025, Ottawa, ON, Canada, April 28-29, 2025*. IEEE, 136–147. doi:10.1109/AST66626.2025.00020
[91] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. *CoRR* abs/2302.13971 (2023).
[92] Jin Wang, Zishan Huang, Hengli Liu, Nianyi Yang, and Yinhao Xiao. 2023. DefectHunter: A Novel LLM-Driven Boosted-Conformer-based Code Vulnerability Detection Mechanism. *CoRR* abs/2309.15324 (2023).
[93] Kuan Wang, Yadong Lu, Michael Santacroce, Yeyun Gong, Chao Zhang, and Yelong Shen. 2023. Adapting LLM Agents Through Communication. *CoRR* abs/2310.01444 (2023).
[94] Weizhe Wang, Wei Ma, Qiang Hu, Yao Zhang, Jianfei Sun, Bin Wu, Yang Liu, Guangquan Xu, and Lingxiao Jiang. 2025. VulnRepairEval: An Exploit-Based Evaluation Framework for Assessing Large Language Model Vulnerability Repair Capabilities. *arXiv preprint arXiv:2509.03331* (2025).
[95] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, and Denny Zhou. 2022. Rationale-Augmented Ensembles in Language Models. *CoRR* abs/2207.00747 (2022).
[96] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *NeurIPS*.
[97] Dumidu Wijayasekara, Milos Manic, and Miles McQueen. 2014. Vulnerability identification and classification via text mining bug databases. In *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 3612–3618.
[98] Dumidu Wijayasekara, Milos Manic, Jason L. Wright, and Miles McQueen. 2012. Mining Bug Databases for Unidentified Software Vulnerabilities. In *2012 5th International Conference on Human System Interactions*. IEEE, 89–96.
[99] Fang Wu, Jigang Wang, Jiqiang Liu, and Wei Wang. 2017. Vulnerability detection with deep learning. In *2017 3rd IEEE international conference on computer and communications (ICCC)*. IEEE, 1298–1302.
[100] Susheng Wu, Wenyan Song, Kaifeng Huang, Bihuan Chen, and Xin Peng. 2024. Identifying Affected Libraries and Their Ecosystems for Open Source Software Vulnerabilities. In *ICSE'24*. ACM, 162:1–162:12.
[101] Yueming Wu, Deqing Zou, Shihan Dou, Siru Yang, Wei Yang, Feng Cheng, Hong Liang, and Hai Jin. 2020. SCDetector: Software Functional Clone Detection Based on Semantic Tokens Analysis. In *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020*. IEEE, 821–833.
[102] Fabian Yamaguchi, Christian Wressnegger, Hugo Gascon, and Konrad Rieck. 2013. Chucky: exposing missing checks in source code for vulnerability discovery. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13*. ACM, 499–510.
[103] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang,

Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. 2024. Qwen2 Technical Report. *CoRR* abs/2407.10671 (2024). arXiv:2407.10671 doi:10.48550/ARXIV.2407.10671

[104] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations, ICLR 2023*. OpenReview.net.

[105] Kang Min Yoo, Dongju Park, Jaewook Kang, Sang-Woo Lee, and Woo-Myoung Park. 2021. GPT3Mix: Leveraging Large-scale Language Models for Text Augmentation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*. Association for Computational Linguistics, 2225–2239.

[106] Hao Yu, Xing Hu, Ge Li, Ying Li, Qianxiang Wang, and Tao Xie. 2022. Assessing and Improving an Evaluation Dataset for Detecting Semantic Code Clones via Deep Learning. *ACM Trans. Softw. Eng. Methodol.* 31, 4 (2022), 62:1–62:25.

[107] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *8th International Conference on Learning Representations, ICLR 2020*. OpenReview.net.

[108] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Kaixuan Wang, and Xudong Liu. 2019. A novel neural source code representation based on abstract syntax tree. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019*. IEEE / ACM, 783–794.

[109] Yangheng Zhao, Zhen Xiang, Sheng Yin, Xianghe Pang, Yanfeng Wang, and Siheng Chen. 2024. MADE: Malicious Agent Detection for Robust Multi-Agent Collaborative Perception. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2024, Abu Dhabi, United Arab Emirates, October 14-18, 2024*. IEEE, 13817–13823. doi:10.1109/IROS58592.2024.10801337

[110] Sipeng Zheng, Jiazheng Liu, Yicheng Feng, and Zongqing Lu. 2023. Steve-Eye: Equipping LLM-based Embodied Agents with Visual Perception in Open Worlds. *CoRR* abs/2310.13255 (2023).

[111] Yunhui Zheng, Saurabh Pujar, Burn L. Lewis, Luca Buratti, Edward A. Epstein, Bo Yang, Jim Laredo, Alessandro Morari, and Zhong Su. 2021. D2A: A Dataset Built for AI-Based Vulnerability Detection Methods Using Differential Analysis. In *ICSE (SEIP)'21*. IEEE, 111–120.

[112] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large Language Models are Human-Level Prompt Engineers. In *The Eleventh International Conference on Learning Representations, ICLR 2023*. OpenReview.net.

[113] Yuhui Zhu, Guanjun Lin, Lipeng Song, and Jun Zhang. 2023. The application of neural network for software vulnerability detection: a review. *Neural Comput. Appl.* 35, 2 (2023), 1279–1301.

Received 26 December 2024; revised 7 January 2026; accepted 4 March 2026