Large Language Models for Cyber Security: A Systematic Literature Review

HANXIANG XU, Huazhong University of Science and Technology, China
SHENAO WANG, Huazhong University of Science and Technology, China
NINGKE LI, Huazhong University of Science and Technology, China
KAILONG WANG*, Huazhong University of Science and Technology, China
YANJIE ZHAO, Huazhong University of Science and Technology, China
KAI CHEN*, Huazhong University of Science and Technology, China
TING YU, Hamad Bin Khalifa University, The State of Qatar
YANG LIU, Nanyang Technological University, Singapore
HAOYU WANG*, Huazhong University of Science and Technology, China

The rapid advancement of Large Language Models (LLMs) has opened up new opportunities for leveraging artificial intelligence in a variety of application domains, including cybersecurity. As the volume and sophistication of cyber threats continue to grow, there is an increasing need for intelligent systems that can automatically detect vulnerabilities, analyze malware, and respond to attacks. In this survey, we conduct a comprehensive review of the literature on the application of LLMs in cybersecurity (LLM4Security). By comprehensively collecting over 40K relevant papers and systematically analyzing 185 papers from top security and software engineering venues, we aim to provide a holistic view of how LLMs are being used to solve diverse problems across the cybersecurity domain.

Through our analysis, we identify several key findings. First, we observe that LLMs are being applied to an expanding range of cybersecurity tasks, including vulnerability detection, malware analysis, and network intrusion detection. Second, we analyze application trends of different LLM architectures (such as encoder-only, encoder-decoder, and decoder-only) across security domains. Third, we identify increasingly sophisticated techniques for adapting LLMs to cybersecurity, such as advanced fine-tuning, prompt engineering, and external augmentation strategies. A significant emerging trend is the use of LLM-based autonomous agents, which represent a paradigm shift from single-task execution to orchestrating complex, multi-step security workflows. Furthermore, we find that the datasets used for training and evaluating LLMs are often limited, highlighting the need for more comprehensive datasets and the use of LLMs for data augmentation. Finally, we discuss the main challenges and opportunities for future research, including the need for more interpretable models, addressing the inherent security risks of LLMs, and their potential for proactive defense.

Overall, our survey provides a comprehensive overview of the current state-of-the-art in LLM4Security and identifies several promising directions for future research. We believe that the insights and findings presented in this survey will contribute to the growing body of knowledge on the application of LLMs in cybersecurity and provide valuable guidance for researchers and practitioners working in this field.

1 INTRODUCTION

The rapid advancements in natural language processing (NLP) over the past decade have been largely driven by the development of large language models (LLMs). By leveraging the Transformer architecture [288] and training on

*Corresponding authors: {wangkl, kchen, haoyuwang}@hust.edu.cn

Authors' addresses: Hanxiang Xu, Huazhong University of Science and Technology, China; Shenao Wang, Huazhong University of Science and Technology, China; Ningke Li, Huazhong University of Science and Technology, China; Kailong Wang, Huazhong University of Science and Technology, China; Yanjie Zhao, Huazhong University of Science and Technology, China; Kai Chen, Huazhong University of Science and Technology, China; Ting Yu, Hamad Bin Khalifa University, The State of Qatar; Yang Liu, Nanyang Technological University, Singapore; Haoyu Wang, Huazhong University of Science and Technology, China.

massive amounts of textual data, LLMs like BERT [65], GPT-3,4 [217, 219], PaLM [55], Claude [18] and Chinchilla [111] have achieved remarkable performance across a wide range of NLP tasks, including language understanding, generation, and reasoning. These foundational models learn rich linguistic representations that can be adapted to downstream applications with minimal fine-tuning, enabling breakthroughs in domains such as open-domain question answering [2], dialogue systems [221, 333], and program synthesis [6].

In particular, one important domain where LLMs are beginning to show promise is cybersecurity. With the growing volume and sophistication of cyber threats, there is an urgent need for intelligent systems that can automatically detect vulnerabilities, analyze malware, and respond to attacks [23, 47, 202]. Recent research has explored the application of LLMs across a wide range of cybersecurity tasks, i.e., **LLM4Security** hereafter. In the domain of software security, LLMs have been used for detecting vulnerabilities from natural language descriptions and source code, as well as generating security-related code, such as patches and exploits. These models have shown high accuracy in identifying vulnerable code snippets and generating effective patches for common types of vulnerabilities [38, 54, 90]. Beyond code-level analysis, LLMs have also been applied to understand and analyze higher-level security artifacts, such as security policies and privacy policies, helping to classify documents and detect potential violations [105, 198]. In the realm of network security, LLMs have demonstrated the ability to detect and classify various types of attacks from network traffic data, including DDoS attacks, port scanning, and botnet traffic [12, 13, 205]. Malware analysis is another key area where LLMs are showing promise, with models being used to classify malware families based on textual analysis reports and behavioral descriptions, as well as detecting malicious domains and URLs [134, 178].

LLMs have also been employed in the field of social engineering to detect and defend against phishing attacks by analyzing email contents and identifying deceptive language patterns [128, 243]. Moreover, researchers are exploring the use of LLMs to enhance the robustness and resilience of security systems themselves, by generating adversarial examples for testing the robustness of security classifiers and simulating realistic attack scenarios for training and evaluation purposes [39, 250, 278]. These diverse applications demonstrate the significant potential of LLMs to improve the efficiency and effectiveness of cybersecurity practices by processing and extracting insights from large amounts of unstructured text, learning patterns from vast datasets, and generating relevant examples for testing and training purposes.

While there have been several valuable efforts in the literature to survey the LLM4Security [58, 66, 206, 332], given the growing body of work in this direction, these studies often have a more focused scope. Many of the existing surveys primarily concentrate on reviewing the types of tasks that LLMs can be applied to, without providing an extensive analysis of other essential aspects related to these tasks, such as the data and domain-specific techniques employed [215, 334], as shown in Table 1. For example, Divakaran et al. [66] only analyzed the prospects and challenges of LLMs in various security tasks, discussing the characteristics of each task separately. However, it lacks insight into the connection between the requirements of these security tasks and data, as well as the application of LLMs in domain-specific technologies.

Scope. To address these limitations and provide an in-depth understanding of the state-of-the-art in LLM4Security, we conduct a systematic and extensive survey of the literature. By comprehensively collecting 47,135 relevant papers and systematically analyzing 185 papers from top security and software engineering venues, our survey aims to provide a holistic view of how LLMs are being applied to solve diverse problems across the cybersecurity domain. In addition to identifying the types of tasks that LLMs are being used for, we also examine the specific datasets, preprocessing techniques, and domain adaptation methods employed in each case. This enables us to provide a more nuanced analysis

Table 1. State-of-the-art surveys related to LLMs for security.

Reference	Year	Scope of topics	Dimensions of discourse	Papers
Motlagh et al. [113]	2024	Security application	Task	Not specified
Divakaran et al [66]	2024	Security application	Task	Not specified
Yao et al. [332]	2024	Security application Security of LLM	Model Task	281
Yigit et al. [334]	2024	Security application Security of LLM	Task	Not specified
Coelho et al. [58]	2024	Security application	Task Domain specific technique	19
Novelli et al. [215]	2024	Security application Security of LLM	Task	Not specified
LLM4Security	2024	Security application	**Model** **Task** **Domain specific technique** **Data**	185

of the strengths and limitations of different approaches, and to identify the most promising directions for future research. Specifically, we focus on answering four key research questions (RQs):

- RQ1: What types of security tasks have been facilitated by LLM-based approaches?
- RQ2: What LLMs have been employed to support security tasks?
- RQ3: What domain specification techniques are used to adapt LLMs to security tasks?
- RQ4: What is the difference in data collection and pre-processing when applying LLMs to various security tasks?

For each research question, we provide a fine-grained analysis of the approaches, datasets, and evaluation methodologies used in the surveyed papers. We identify common themes and categorize the papers along different dimensions to provide a structured overview of the landscape. Furthermore, we highlight the key challenges and limitations of current approaches to guide future research towards addressing the gaps.

Significance. A key significance of this survey lies in its deep-seated connection to Software Engineering (SE) community. This connection is not incidental; our systematic selection process deliberately includes premier SE venues (e.g., ICSE, FSE, ASE, etc.), and our analysis reveals that the vast majority of research (63%) falls under the umbrella of software and system security. The surveyed applications of LLMs are fundamentally addressing core challenges within the secure software development lifecycle, ranging from vulnerability detection and automated program repair to fuzzing and secure code generation. More profoundly, we observe that the methodologies employed in LLM4Security represent an evolution of established SE research paradigms [113]. For instance, the generative capabilities of LLMs are pushing the boundaries of classic SE tasks like automated testing and program repair, while the reliance on curated code corpora and the novel use of LLMs for data augmentation align directly with the principles of data-driven software engineering. Therefore, this survey not only maps a subfield of cybersecurity but also chronicles a significant, ongoing transformation within software engineering itself.

Contributions. The contributions of this work are summarized as follows:

- We provide the first comprehensive and structured map of the LLM4Security landscape, synthesized from a Systematic Literature Review (SLR) of 185 high-quality papers selected from an initial pool of over 47,000 candidates.
- We deliver a multi-dimensional analysis of the field by systematically investigating four crucial aspects across the selected studies: the security tasks being automated, the specific LLM architectures being employed, the domain-specific techniques (e.g., fine-tuning and external augmentation) used for adaptation, and the data-centric practices for training and evaluation.
- We synthesize and present key insights into how LLMs are reshaping the Secure-Software-Engineering lifecycle. Our findings reveal a paradigm shift from mere defect detection towards automated generation and repair; the use of LLMs for data augmentation to tackle data scarcity; and the emergence of hybrid workflows where LLMs orchestrate traditional SE tools, heralding a new era of security automation.

The survey progresses with the following framework. We outline our survey methodology, including the search strategy, inclusion/exclusion criteria, and the data extraction process, in Section 2. The analysis and findings for each of the four research questions can be found in Sections 3 through 6. Sections 7 to 8 explore the constraints and significance of our results, while also identifying promising directions for future research. Finally, Section 9 concludes the paper.

2 METHODOLOGY

In this study, we conducted a **Systematic Literature Review (SLR)** to investigate the latest research on **LLM4Security**. This review aims to provide a comprehensive mapping of the landscape, identifying how LLMs are being deployed to enhance cybersecurity measures.

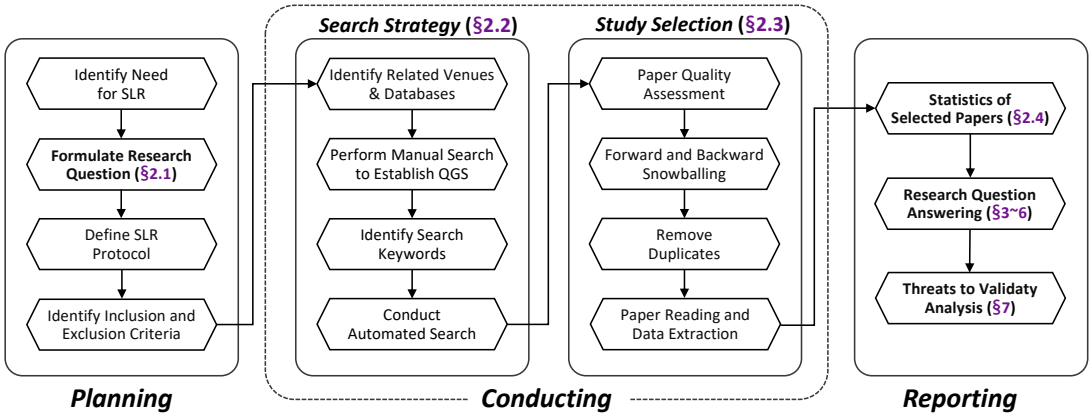


Fig. 1. Systematic Literature Review Methodology for LLM4Security.

Following the established SLR guidelines [144, 233], our methodology is structured into three pivotal stages as shown in Figure 1: Planning (§2.1), Conducting (§2.2, §2.3), and Reporting (§2.4), each meticulously designed to ensure comprehensive coverage and insightful analysis of the current state of research in this burgeoning field.

Planning. Initially, we formulated precise research questions to understand how LLMs are being utilized in security tasks, the benefits derived, and the associated challenges. Subsequently, we developed a detailed protocol delineating

our search strategy, including specific venues and databases, keywords, and quality assessment criteria. Each co-author reviewed this protocol to enhance its robustness and align with our research objectives.

Literature survey and analysis. We meticulously crafted our literature search to ensure comprehensiveness, employing both manual and automated strategies across various databases to encompass a wide range of papers. Each study identified underwent a stringent screening process, initially based on their titles and abstracts, followed by a thorough review of the full text to ensure conformity with our predefined criteria. To prevent overlooking related papers, we also conducted forward and backward snowballing on the collected papers.

Reporting. We present our findings through a structured narrative, complemented by visual aids like flowcharts and tables, providing a clear and comprehensive overview of the existing literature. The discussion delves into the implications of our findings, addressing the potential of LLMs to revolutionize cybersecurity practices and identifying gaps that warrant further investigation.

2.1 Research Question

The primary aim of this SLR, focused on the context of LLM4Security, is to meticulously dissect and synthesize existing research at the intersection of these two critical fields. This endeavor seeks to illuminate the multifaceted applications of LLMs in cybersecurity, assess their effectiveness, and delineate the spectrum of methodologies employed across various studies. To further refine this objective, we formulated the following four **Research Questions (RQs)**:

- **RQ1: What types of security tasks have been facilitated by LLM-based approaches?** Here, the focus is on the scope and nature of security tasks that LLMs have been applied to. The goal is to categorize and understand the breadth of security challenges that LLMs are being used to address, highlighting the model's adaptability and effectiveness across various security dimensions. We will categorize previous studies according to different security domains and provide detailed insights into the diverse security tasks that use LLMs in each security domain.
- **RQ2: What LLMs have been employed to support security tasks?** This RQ seeks to inventory the specific LLMs that have been utilized in security tasks. Understanding the variety and characteristics of LLMs used can offer insights into their versatility and suitability for different security applications. We will discuss the architectural differences of LLMs and delve into analyzing the impact of LLMs with different architectures on cybersecurity research over different periods.
- **RQ3: What domain specification techniques are used to adapt LLMs to security tasks?** This RQ delves into the specific methodologies and techniques employed to fine-tune or adapt LLMs for security tasks. Understanding these techniques can provide valuable insights into the customization processes that enhance LLMs' effectiveness in specialized tasks. We will elucidate how LLMs are applied to security tasks by analyzing the domain-specific techniques employed in papers, uncovering the inherent and specific connections between these techniques and particular security tasks.
- **RQ4: What is the difference in data collection and pre-processing when applying LLMs to security tasks?** This RQ aims to explore the unique challenges and considerations in data processing and model evaluation within the security environment, investigating the correlation between LLMs and the data used for specific tasks. We will reveal the challenges arising from data in applying LLMs to security tasks through two dimensions: data collection and data preprocessing. Additionally, we will summarize the intrinsic relationship among data, security tasks, and LLMs.

2.2 Search Strategy

To collect and identify a set of relevant literature as accurately as possible, we employed the "Quasi-Gold Standard" (QGS) [345] strategy for literature search. The overview of the strategy we applied in this work is as follows:

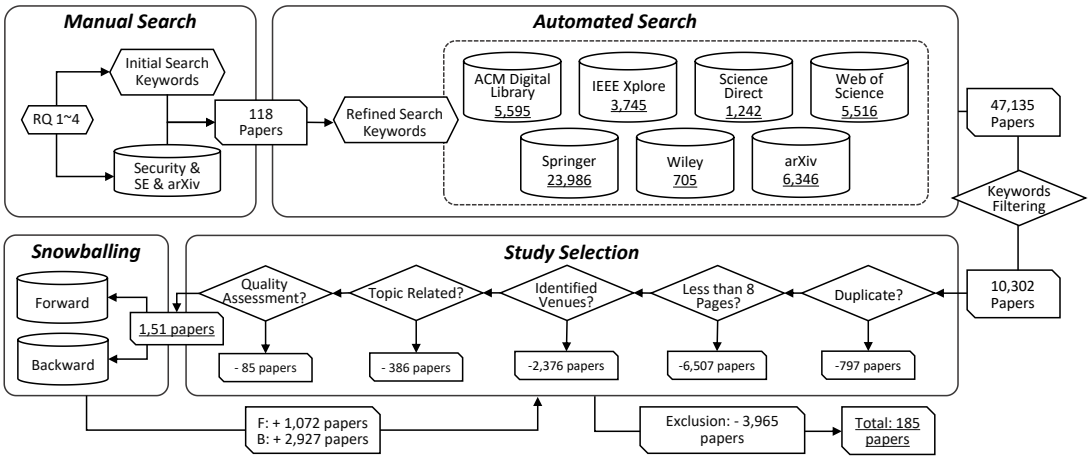


Fig. 2. Paper Search and Selection Process.

Step1: Identify related venues and databases. To initiate this approach, we first identify specific venues for manual search and then choose suitable libraries and databases for the automated search. In this stage, we opt for six of the top Security conferences and journals (i.e., S&P, NDSS, USENIX Security, CCS, TDSC, and TIFS) as well as six of the leading Software Engineering conferences and journals (i.e.,ICSE, ESEC/FSE, ISSTA, ASE, TOSEM, and TSE). Given the emerging nature of LLMs in research, we also include arXiv in both manual and automated searches, enabling us to capture the latest unpublished studies in this rapidly evolving field. For automated searches, we select seven widely utilized databases, namely the ACM Digital Library, IEEE Xplore, Science Direct, Web of Science, Springer, Wiley, and arXiv. These databases offer comprehensive coverage of computer science literature and are commonly employed in systematic reviews within this domain [113, 340, 364].

Step2: Establish QGS. In this step, we start by creating a manually curated set of studies that have been carefully screened to form the QGS. A total of 118 papers relevant to LLM4Sec are manually identified, aligning with the research objective and encompassing various techniques, application domains, and evaluation methods.

Step3: Define search keywords. The keywords for automatic search are elicited from the title and abstract of the selected QGS papers through word frequency analysis. The search string consists of two sets of keywords:

- *Keywords related to LLM: Large Language Model, LLM, Language Model, LM, Pre-trained, CodeX, Llama, GPT-*, ChatGPT, T5, AIGC, AGI.*
- *Keywords related to Security tasks: Cyber Security, Web Security, Network Security, System Security, Software Security, Data Security, Program Analysis, Program Repair, Software Vulnerability, CVE, CWE, Vulnerability Detection, Vulnerability Localization, Vulnerability Classification, Vulnerability Repair, Software Bugs, Bug Detection, Bug Localization, Bug Classification, Bug Report, Bug Repair, Security Operation, Privacy Violation, Denial of Service, Data Poison, Backdoor, Malware Detection, Malware Analysis, Ransomware, Malicious Command, Fuzz Testing, Penetration Testing, Phishing, Fraud, Scam, Forensics, Intrusion Detection.*

Table 2. Inclusion and exclusion criteria.

Inclusion Criteria
In#1: The title and abstract of the paper contain a pair of identified search keywords;
In#2: Papers that apply large language models (e.g., BERT, GPT, T5) to security tasks;
In#3: Papers that propose new techniques or models for security tasks based on large language models;
In#4: Papers that evaluate the performance or effectiveness of large language models in security contexts.
Exclusion Criteria
Ex#1: Duplicate papers, studies with little difference in multi-version from the same authors;
Ex#2: Short papers less than 8 pages, tool demos, keynotes, editorials, books, thesis, workshop papers, or poster papers;
Ex#3: Papers not published in identified conferences or journals, nor as preprints on arXiv;
Ex#4: Papers that do not focus on security tasks (e.g., natural language processing tasks in general domains);
Ex#5: Papers that use traditional machine learning or deep learning techniques without involving large language models;
Ex#6: Secondary studies, such as an SLR, review, or survey;
Ex#7: Papers not written in English;
Ex#8: Papers focus on the security of LLMs rather than using LLMs for security tasks.

Step4: Conduct an automated search. These identified keywords are paired one by one and input into automated searches across the above-mentioned seven widely used databases. Our automated search focused on papers published after 2019, in which GPT-2 was published, as it marked a significant milestone in the development of large language models. The search was conducted in the title, abstract, and keyword fields of the papers in each database. Specifically, the number of papers retrieved from each database after applying the search query and the year filter (2019-2025) is as follows: 5,595 papers in ACM Digital Library, 3,745 papers in IEEE Xplore, 1,242 papers in Science Direct, 5,516 papers in Web of Science, 23,986 papers in Springer, 705 papers in Wiley, and 6,346 papers in arXiv.

2.3 Study Selection

After obtaining the initial pool of 47,135 papers (47,017 from the automated search and 118 from the QGS), we conducted a multi-stage study selection process to identify the most relevant and high-quality papers for our systematic review.

2.3.1 Coarse-Grained Inclusion and Exclusion Criteria. To select relevant papers for our research questions, we defined four inclusion criteria and eight exclusion criteria (as listed in Table 2) for the coarse-grained paper selection process. Among them, In#1, Ex#1, Ex#2, and Ex#3 were automatically filtered based on the keywords, duplication status, length, and publication venue of the papers. The remaining inclusion criteria (In#2~4) and exclusion criteria (Ex#4~8) were manually applied by inspecting the topic and content of each paper. Specifically, the criteria of In#1 retained 10,302 papers whose titles and abstracts contained a pair of the identified search keywords. Subsequently, Ex#1 filtered out 797 duplicate or multi-version papers from the same authors with little difference. Next, the automated filtering criteria Ex#2 was applied to exclude short papers, tool demos, keynotes, editorials, books, theses, workshop papers, or poster papers, resulting in 6,507 papers being removed. The remaining papers were then screened based on the criteria Ex#3, which retained 622 full research papers published in the identified venues or as preprints on arXiv. The remaining inclusion and exclusion criteria (In#2~4, Ex#4~8) were then manually applied to the titles and abstracts of these 622 papers, in order to determine their relevance to the research topic. Three researchers independently applied the inclusion and

exclusion criteria to the titles and abstracts. Disagreements were resolved through discussion and consensus. After this manual inspection stage, 236 papers were included for further fine-grained full-text quality assessment.

2.3.2 Fine-grained Quality Assessment. To ensure the included papers are of sufficient quality and rigor, we assessed them using a set of quality criteria adapted from existing guidelines for systematic reviews in software engineering. The quality criteria included:

- **QAC#1:** Clarity and appropriateness of research goals and questions;
- **QAC#2:** Adequacy of methodology and study design;
- **QAC#3:** Rigor of data collection and analysis processes;
- **QAC#4:** Validity of results and conclusions;
- **QAC#5:** Thoroughness of reporting and documentation.

Each criterion was scored on a 3-point scale (0: not met, 1: partially met, 2: fully met). Papers with a total score of 6 or higher (out of 10) were considered as having acceptable quality. After the quality assessment, 151 papers remained in the selected set.

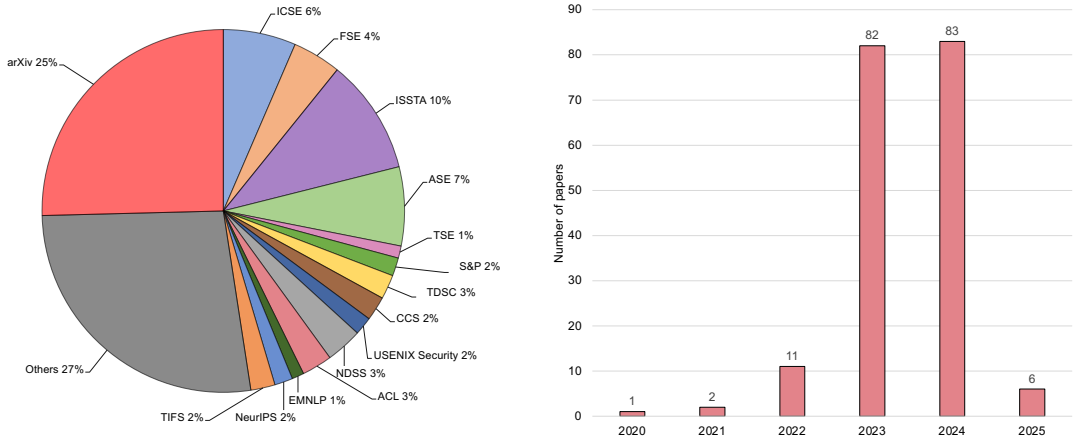
2.3.3 Forward and Backward Snowballing. To further expand the coverage of relevant literature, we performed forward and backward snowballing on the 151 selected papers. Forward snowballing identified papers that cited the selected papers, while backward snowballing identified papers that were referenced by the selected papers.

Here we obtained 1,072 and 2,927 papers separately during the forward and backward processes. Then we applied the same inclusion/exclusion criteria and quality assessment to the papers found through snowballing. After the initial keyword filtering and deduplication, there were 1,583 papers that remained available. Among them, 77 papers were excluded during the page number filtering step, and 1,472 papers were deleted to ensure the papers were published in the selected venues. After confirming the paper topics and assessing the paper quality, only 34 papers were ultimately retained in the snowballing process, resulting in a final set of 185 papers for data extraction and synthesis.

2.4 Statistics of Selected Papers

After conducting searches and snowballing, a total of 185 relevant research papers were ultimately obtained. The distribution of the included documents is outlined in Figure 3. As depicted in Figure 3(A), approximately 75% of the papers are from peer-reviewed venues or other sources, while the remaining 25% were published on arXiv, an open-access platform serving as a repository for scholarly articles. Among the specified peer-reviewed venues, ISSTA had the highest frequency (10%), followed by ICSE (8%) and ASE (7%). Other venues making significant contributions included FSE (4%), NDSS (3%), ACL (3%), TDSC (3%), S&P (2%), CCS (2%), USENIX Security (2%), TIFS (2%), NeurIPS (2%), EMNLP (1%), and TSE (1%). The significant portion from arXiv is unsurprising given the rapid emergence of new LLM4Security studies, with many works recently completed and potentially undergoing peer review. Despite lacking peer review in some cases, we conducted rigorous quality assessments on all collected papers to ensure the integrity of our investigation results. This approach enables us to include all high-quality and relevant publications while upholding stringent research standards.

The temporal distribution of the included papers is depicted in Figure 3(B). Since 2020, there has been a notable upward trend in the number of publications. In 2020, only 1 relevant paper was published, followed by 2 in 2021, and 11 papers in 2022. Subsequently, the field experienced explosive growth, with the total count surging to 82 papers in 2023 and a similar peak of 83 papers in 2024. This rapid growth trend signifies an increasing interest in LLM4Security



(A) Distribution of papers across venues.

(B) Distribution of papers over years.

Fig. 3. Overview of the selected 185 papers' distribution.

Table 3. Extracted data items and related research questions (RQs).

RQ	Data Item
1,2,3,4	The category of LLM
1,3,4	The category of cybersecurity domain
1,2,3	Attributes and suitability of LLMs
1,3	Security task requirements and the application of LLM solutions
1	The security task to which the security domain belongs
3	Techniques to adapt LLMs to tasks
3	Prominent external enhancement techniques
4	The types and features of datasets used

research. Our review also includes 6 early papers from 2025, reflecting the ongoing momentum. We will continue to observe the developments in LLM4Security research.

After completing the full-text review phase, we proceeded with data extraction. The objective was to collect all relevant information essential for offering detailed and insightful answers to the RQs outlined in §2.1. As illustrated in Table 3, the extracted data included the categorization of security tasks, their corresponding domains, as well as classifications of LLMs, external enhancement techniques, and dataset characteristics. Using the gathered data, we systematically examined the relevant aspects of LLM application within the security domains.

3 RQ1: WHAT TYPES OF SECURITY TASKS HAVE BEEN FACILITATED BY LLM-BASED APPROACHES?

This section delves into the detailed examination of LLM utilization across diverse security domains. We have classified them into six primary domains, aligning with the themes of the collected papers: software and system security, network security, information and content security, hardware security, and blockchain security, totaling 185 papers.

Figure 4 visually depicts the distribution of LLMs within these six domains. Additionally, Table 4 offers a comprehensive breakdown of research detailing specific security tasks addressed through LLM application. We also analyze the recent and significant trend of employing LLM-based autonomous agents, which represent a paradigm shift from performing isolated tasks to orchestrating complex security workflows.

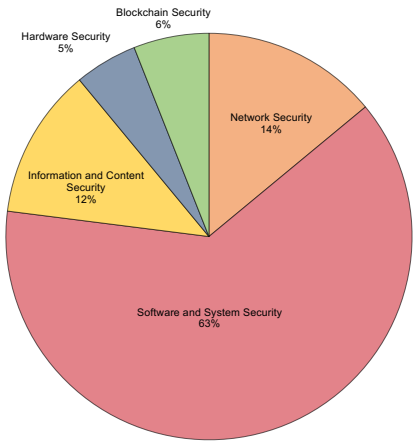


Fig. 4. Distribution of LLM usages in security domains.

The majority of research activity in the realm of software and system security, constituting around 63% of the total research output, is attributed to the advancements made by code LLMs [249, 355, 360] and the extensive applications of LLMs in software engineering [113]. This emphasis underscores the significant role and impact of LLMs in software and system security, indicating a predominant focus on leveraging LLMs to automate the handling of potential security issues in programs and systems. Approximately 14% of the research focus pertains to network security tasks, highlighting the importance of LLMs in aiding traffic detection and network threat analysis. Information and content security activities represent around 12% of the research output, signaling a growing interest in employing LLMs for generating and detecting fake content. Conversely, activities in hardware security and blockchain security account for approximately 5% and 6% of the research output, respectively, suggesting that while exploration in these domains has been comparatively limited thus far, there remains research potential in utilizing LLMs to analyze hardware-level vulnerabilities and potential security risks in blockchain technology.

3.1 Application of LLMs in Network Security

This section explores the application of LLMs in the field of network security. The tasks include web fuzzing, intrusion and anomaly detection, cyber threat analysis, and penetration testing.

Web fuzzing. Web fuzzing is a mutation-based fuzzer that generates test cases incrementally based on the coverage feedback it receives from the instrumented web application [286]. Security is undeniably the most critical concern for web applications. Fuzzing can help operators discover more potential security risks in web applications. Liang et al. [166] proposed GPTFuzzer based on a decoder-only architecture. It generates effective payloads for web application firewalls (WAFs) targeting SQL injection, XSS, and RCE attacks by generating fuzz test cases. The model undergoes reinforcement learning [160] fine-tuning and KL-divergence penalty to effectively generate attack payloads and mitigate

Table 4. Distribution of security tasks over six security domains.

Security Domains	Security Tasks	Total
Network Security	Web fuzzing (3) Traffic and intrusion detection (10) Cyber threat analysis (5) Penetration test (4) Protocol analysis (1) Protocol fuzzing (2) Web vulnerability detection (1)	26
Software and System Security	Vulnerability detection (22) Vulnerability repair (15) Bug detection (11) Bug repair (32) Program fuzzing (11) Reverse engineering and binary analysis (10) Malware detection (3) System log analysis (10) Secure code generation (2) ADS security verification (2) Code obfuscation (1)	119
Information and Content Security	Phishing and scam detection (8) Harmful contents detection (8) Steganography (2) Access control (1) Forensics (1)	20
Hardware Security	Hardware vulnerability detection (2) Hardware vulnerability repair (5) Hardware IP protection (1) Security assertions generation (1)	9
Blockchain Security	Smart contract vulnerability detection (10) Transaction anomaly detection (1)	11

the local optimum issue. Similarly, Liu et al. [175] utilized an encoder-decoder architecture model to generate SQL injection detection test cases for web applications, enabling the translation of user inputs into new test cases. Meng et al.'s CHATAFL [195], on the other hand, shifts focus to leveraging LLMs for generating structured and sequenced effective test inputs for network protocols lacking machine-readable versions.

Traffic and intrusion detection. Detecting network traffic and intrusions is a crucial aspect of network security and management [201]. LLMs have been widely applied in network intrusion detection tasks, covering traditional web applications, IoT (Internet of Things), and in-vehicle network scenarios [13, 85, 193, 202]. LLMs not only learn the characteristics of malicious traffic data [12, 13, 202] and capture anomalies in user-initiated behaviors [29] but also describe the intent of intrusions and abnormal behaviors [3, 12, 80]. Additionally, they can provide corresponding security recommendations and response strategies for identified attack types [48]. Liu et al. [178] proposed a method for detecting malicious URL behavior by utilizing LLMs to extract hierarchical features of malicious URLs. Their work extends the application of LLMs in intrusion detection tasks to the user level, demonstrating the generality and effectiveness of LLMs in intrusion and anomaly detection tasks.

Cyber threat analysis. In contemporary risk management strategies, Cyber Threat Intelligence (CTI) reporting plays a pivotal role, as evidenced by recent research [45]. With the continued surge in the volume of CTI reports, there

is a growing need for automated tools to facilitate report generation. The application of LLMs in network threat analysis can be categorized into CTI generation and CTI analysis for decision-making. The emphasis on CTI generation varies, including extracting CTI from network security text information (such as books, blogs, news) [5], generating structured CTI reports from unstructured information [264], and generating CTI from network security entity graphs [231]. Aghaei et al.'s CVEDrill [4] can generate priority recommendation reports for potential cybersecurity threats and predict their impact. Additionally, Moskal et al. [205] explored the application of ChatGPT in assisting or automating response decision-making for threat behaviors, demonstrating the potential of LLMs in addressing simple network attack activities.

Penetration test. Conducting a controlled attack on a computer system to evaluate its security is the essence of penetration testing, which remains a pivotal approach utilized by organizations to bolster their defenses against cyber threats [256]. The general penetration testing process consists of three steps: information gathering, payload construction, and vulnerability exploitation. Temara [278] utilized LLMs to gather information for penetration testing, including the IP address, domain information, vendor technologies, SSL/TLS credentials, and other details of the target website. Sai Charan et al. [39] critically examined the capability of LLMs to generate malicious payloads for penetration testing, with results indicating that ChatGPT can generate more targeted and complex payloads for attackers. Happe et al. [104] developed an automated Linux privilege escalation guidance tool using LLMs. Additionally, the automated penetration testing tool PentestGPT [61], based on LLMs, achieved excellent performance on a penetration testing benchmark containing 13 scenarios and 182 subtasks by combining three self-interacting modules (inference, generation, and parsing modules).

Protocol analysis. LLMs are being applied to analyze complex network protocol specifications for detecting inconsistencies. For example, CellularLint utilizes domain-adapted LLMs combined with few-shot and active learning techniques to systematically identify behavioral inconsistencies in 4G/5G cellular standards [242]. Such inconsistencies found in specifications can impact the security and interoperability of network implementations.

Protocol fuzzing. LLMs are utilized to guide protocol fuzzing, particularly for protocols specified only in natural language, overcoming limitations of traditional fuzzers in generating structurally and sequentially valid inputs. Approaches like ChatAFL integrate LLMs into the fuzzing loop, using the model's understanding of message types and sequences to generate valid test inputs and guide mutation-based fuzzers like AFL++ [196]. Other frameworks, such as LLMIF, augment LLMs with protocol specifications to automatically extract message formats, field constraints, and state transition rules, and to reason about device responses during fuzzing, specifically targeting IoT protocols [297]. These LLM-guided approaches have demonstrated significant improvements in code coverage and vulnerability discovery compared to baseline fuzzers.

Web vulnerability detection. LLMs are also employed for detecting vulnerabilities in web applications, although challenges exist regarding language-specific characteristics and data availability. Frameworks like RealVul aim to enhance the performance of LLMs for detecting PHP vulnerabilities [35]. This is achieved by integrating deep program analysis techniques, such as control and data flow analysis, to identify vulnerability candidates and extract relevant code slices.

3.2 Application of LLMs in Software and System Security

This section explores the application of LLMs in the field of software and system security. LLMs excel in understanding user commands, inferring program control and data flow, and generating complex data structures [310]. The tasks it

includes vulnerability detection, vulnerability repair, bug detection, bug repair, program fuzzing, reverse engineering and binary analysis, malware detection, and system log analysis.

Vulnerability detection. The escalation in software vulnerabilities is evident in the recent surge of vulnerability reports documented by Common Vulnerabilities and Exposures (CVEs) [16]. With this rise, the potential for cybersecurity attacks grows, posing significant economic and social risks. Hence, the detection of vulnerabilities becomes imperative to safeguard software systems and uphold social and economic stability. The method of utilizing LLMs for static vulnerability detection in code shows significant performance improvements compared to traditional approaches based on graph neural networks or matching rules [19, 47, 49, 54, 84, 142, 180, 238, 279, 284, 302, 342, 354]. For instance, SCALE enhances detection by using LLMs to generate comments for Abstract Syntax Tree nodes, creating a novel Structured Natural Language Comment Tree (SCT) representation fed to a graph classifier [307]. Furthermore, techniques like multi-task instruction fine-tuning, as used in VulLLM, aim to improve model generalization by training simultaneously on detection, localization, and vulnerability explanation tasks [69]. The potential demonstrated by GPT series models in vulnerability detection tasks is particularly evident [49, 84, 142, 169, 180, 285, 313, 325, 342]. However, LLMs may generate false positives when dealing with vulnerability detection tasks due to minor changes in function and variable names or modifications to library functions [284]. Liu et al. [176] proposed LATTE, which combines LLMs to achieve automated binary taint analysis. This overcomes the limitations of traditional taint analysis, which requires manual customization of taint propagation rules and vulnerability inspection rules. They discovered 37 new vulnerabilities in real firmware. Tihanyi et al. [281] used LLMs to generate a large-scale vulnerability-labeled dataset, FormAI, while also noting that over 50% of the code generated by LLMs may contain vulnerabilities, posing a significant risk to software security.

Vulnerability repair. Due to the sharp increase in the number of detected vulnerabilities and the complexity of modern software systems, manually fixing security vulnerabilities is extremely time-consuming and labor-intensive for security experts [350]. Research shows that 50% of vulnerabilities have a lifecycle exceeding 438 days [157]. Delayed vulnerability patching may result in ongoing attacks on software systems [172], causing economic losses to users. The T5 model based on the encoder-decoder architecture performs better in vulnerability repair tasks [90, 347]. Although LLMs can effectively generate fixes, challenges remain in maintaining the functionality correctness of functions [227], and they are susceptible to influences from different programming languages [151]. For example, the current capabilities of LLMs in repairing Java vulnerabilities are limited [312], while studies also investigate their effectiveness on JavaScript vulnerabilities, showing performance improvements with better contextual prompts [151]. Constructing a comprehensive vulnerability repair dataset and fine-tuning LLMs on it can significantly improve the model's performance in vulnerability repair tasks [90, 159]. To assist developers, especially when vulnerable code locations are untracked in issue reports, approaches like PATUNTRACK use LLMs with Retrieval-Augmented Generation (RAG) and few-shot learning to generate relevant patch examples (insecure code, patch, explanation) [130]. Another direction involves generating natural language repair suggestions instead of direct code patches; VulAdvisor first uses LLMs to create a dataset of suggestions from code/patch pairs and then fine-tunes CodeT5 with context-aware attention to generate such suggestions directly from vulnerable code [346]. Alrashedy et al. [38] proposed an automated vulnerability repair tool driven by feedback from static analysis tools. Tol et al. [282] proposed a method called ZeroLeak, which utilizes LLMs to repair side-channel vulnerabilities in programs. Charalambous et al. [14] combined LLMs with Bounded Model Checking (BMC) to verify the effectiveness of repair solutions, addressing the problem of decreased functionality correctness after using LLMs to repair vulnerabilities [50].

Bug detection. Bugs typically refer to any small faults or errors present in software or hardware, which may cause programs to malfunction or produce unexpected results. Some bugs may be exploited by attackers to create security vulnerabilities. Therefore, bug detection is crucial for the security of software and system. LLMs can be utilized to generate code lines and compare them with the original code to flag potential bugs within code snippets [7]. They can also combine feedback from static analysis tools to achieve precise bug localization [112, 133, 158]. Fine-tuning techniques are crucial for bug detection tasks as well, applying fine-tuning allows LLMs to identify errors in code without relying on test cases [32, 153, 328]. To address the challenge of hallucinations or false positives in LLM-generated bug reports, frameworks like LLMSAN use techniques such as Chain-of-Thought prompting combined with post-hoc validation of the LLM's claimed evidence (e.g., data-flow paths) via program analysis and targeted queries [290]. Additionally, Du et al. [70] and Li et al. [162] introduced the concept of contrastive learning, which focuses LLMs on the subtle differences between correct and buggy versions of code lines. Fang et al. [78] proposed a software-agnostic representation method called RepresentThemAll, based on contrastive learning and fine-tuning modules, suitable for various downstream tasks including bug detection and predicting the priority and severity of bugs.

Bug repair. LLMs possess robust code generation capabilities, and their utilization in engineering for code generation can significantly enhance efficiency. However, code produced by LLMs often carries increased security risks, such as bugs and vulnerabilities [232]. These program bugs can lead to persistent security vulnerabilities. Hence, automating the process of bug fixing is imperative, involving the use of automation technology to analyze flawed code and generate accurate patches to rectify identified issues. LLMs like CodeBERT [121, 152, 318, 348], CodeT5 [121, 173, 276, 299], Codex [53, 76, 133, 274, 319], LLaMa [276], CodeLLaMa [216, 263] , CodeGEN [319], UniXcoder [348], T5 [338], PLBART [121], Starcoder [357] and GPT Series [82, 123, 216, 276, 295, 319, 320, 329, 336, 348, 349, 351, 353, 362] have showcased effectiveness in generating syntactically accurate and contextually relevant code. This includes frameworks with encoder-decoder architecture like Repilot [305], tailored specifically for producing repair patches. Utilizing LLMs for program repair can achieve competitive performance in producing patches for various types of errors and defects [320]. These models effectively capture the underlying semantics and dependencies in code, resulting in precise and efficient patches. Moreover, fine-tuning LLMs on specific code repair datasets can further improve their ability to generate high-quality patches for real-world software projects. Integrating LLMs into program repair not only speeds up the error-fixing process but also allows software developers to focus on more complex tasks, thereby enhancing the reliability and maintainability of the software [319]. As demonstrated in the case of ChatGPT, notably enhances the accuracy of program repairs when integrated with interactive feedback loops [319]. This iterative process of patch generation and validation fosters a nuanced comprehension of software semantics, thereby resulting in more impactful fixes. Complementary to generating fixes, evaluating the capabilities of LLM-driven code agents requires realistic benchmarks; SWT-Bench, for example, provides a framework based on real-world data to jointly assess test generation and fix validation performance [209]. By integrating domain-specific knowledge and technologies with the capabilities of LLMs, their performance is further enhanced. Custom prompts, fine-tuning for specific tasks, retrieving external data, and utilizing static analysis tools [90, 133, 276, 317, 347] significantly improve the effectiveness of bug fixes driven by LLMs.

Program fuzzing. Fuzz testing, or fuzzing, refers to an automated testing method aimed at generating inputs to uncover unforeseen behaviors. Both researchers and practitioners have effectively developed practical fuzzing tools, demonstrating significant success in detecting numerous bugs and vulnerabilities within real-world systems [25]. The generation capability of LLMs enables testing against various input program languages and different features [62, 89, 316], effectively overcoming the limitations of traditional fuzz testing methods. Under strategies such as repetitive querying,

example querying, and iterative querying [343], LLMs can significantly enhance the generation effectiveness of test cases. LLMs can generate test cases that trigger vulnerabilities from historical bug reports of programs [63], produce test cases similar but different from sample inputs [118], analyze compiler source code to generate programs that trigger specific optimizations [330], and split the testing requirements and test case generation using a dual-model interaction framework, assigning them to different LLMs for processing. LLMs are also used to analyze program documentation to predict high-risk option combinations for targeted fuzzing, as demonstrated by ProphetFuzz [292]. Furthermore, LLMs facilitate the automatic generation of fuzz drivers for libraries [189, 322, 344], for instance, PROMPTFUZZ uses an iterative prompt fuzzing loop guided by code coverage feedback. To improve mutation effectiveness, especially for complex inputs like JavaScript, techniques like CovRL integrate coverage-guided reinforcement learning to fine-tune LLM-based mutators [74].

Reverse engineering and binary analysis. Reverse engineering is the process of attempting to understand how existing artifacts work, whether for malicious purposes or defensive purposes, and it holds significant security implications. The capability of LLMs to recognize software functionality and extract important information enables them to perform certain reverse engineering steps [228]. For example, Xu et al. [326] achieved recovery of variable names from binary files by propagating LLMs query results through multiple rounds. Techniques like GENNM further advance this by using generative LLMs with context-aware fine-tuning, preference optimization, and iterative inference to recover names from stripped binaries [327]. Armengol-Estapé et al. [17] combined type inference engines with LLMs to perform disassembly of executable files and generate program source code. LLMs can also be used to assist in binary program analysis. Sun et al. [270] proposed DexBert for characterizing Android system binary bytecode. Pei et al. [229] preserved the semantic symmetry of code based on group theory, resulting in their binary analysis framework SYMC demonstrating outstanding generalization and robustness in various binary analysis tasks. Song et al. [266] utilized LLMs to address authorship analysis issues in software engineering, effectively applying them to real-world APT malicious software for organization-level verification. Some studies [79, 119, 261] apply LLMs to enhance the readability and usability of decompiler outputs, including decompiling WebAssembly into higher-level languages like C/C++, thereby assisting reverse engineers in better understanding binary files.

Malware detection. Due to the rising volume and intricacy of malware, detecting malicious software has emerged as a significant concern. While conventional detection techniques rely on signatures and heuristics, they exhibit limited effectiveness against unknown attacks and are susceptible to evasion through obfuscation techniques [23]. LLMs can extract semantic features of malware, leading to more competitive performance. AVScan2Vec, proposed by Joyce et al. [134], transforms antivirus scan reports into vector representations, effectively handling large-scale malware datasets and performing well in tasks such as malware classification, clustering, and nearest neighbor search. To address dataset limitations in specific domains like malicious package detection, approaches like Maltracker combine fine-grained code analysis with LLM-generated synthetic malicious samples to create enhanced datasets for training detection models [337]. Botacin [27] explored the application of LLMs in malware defense from the perspective of malware generation. While LLMs cannot directly generate complete malware based on simple instructions, they can generate building blocks of malware and successfully construct various malware variants by blending different functionalities and categories. This provides a new perspective for malware detection and defense.

System log analysis. Analyzing the growing amount of log data generated by software-intensive systems manually is unfeasible due to its sheer volume. Numerous deep learning approaches have been suggested for detecting anomalies in log data. These approaches encounter various challenges, including dealing with high-dimensional and noisy log data, addressing class imbalances, and achieving generalization [127]. Nowadays, researchers are utilizing the language

understanding capabilities of LLMs to identify and analyze anomalies in log data. Studies compare different strategies, such as supervised fine-tuning (SFT) of models versus few-shot in-context learning (ICL) with LLMs, finding SFT often yields higher accuracy given sufficient labeled data, while ICL is advantageous for rapid deployment or limited data scenarios [132]. Compared to traditional deep learning methods, LLMs demonstrate outstanding performance and good interpretability [235, 259]. Fine-tuning LLMs for specific types of logs [129, 141], using reinforcement learning-based fine-tuning strategies [101] can significantly enhance their performance in log analysis tasks. LLMs are also being employed for log analysis in cloud servers [51, 174], where their reasoning abilities can be combined with server logs to infer the root causes of cloud service incidents.

Secure code generation. While LLMs demonstrate strong code generation capabilities, the security of the generated code is a significant concern, as models trained on large corpora may replicate insecure patterns [232]. Studies investigating LLM-generated code that utilizes security APIs, such as Java security APIs, have found high rates of misuse, highlighting the risks of relying on LLMs for security-sensitive implementations [207]. To mitigate these risks, research explores methods to guide LLMs towards more secure code generation; for example, PromSec proposes an interactive loop involving static analysis, graph-based code fixing, and reverse-prompt engineering to automatically optimize prompts for generating functionally correct and more secure code [212].

ADS security verification. LLMs are also being applied to the safety and security verification of complex software-intensive systems like Autonomous Driving Systems (ADS). One application involves leveraging LLMs to enhance simulation testing; for instance, SoVAR utilizes LLMs to automatically extract and generalize test scenarios from natural language accident reports, generating more diverse and realistic test cases for simulation [99]. Another application focuses on automating the analysis of test results; DIAVIO employs fine-tuned LLMs, informed by real-world accident data represented in a domain-specific language, to automatically diagnose the cause and type of safety violations identified during ADS simulation testing [185].

Code obfuscation. Analyzing obfuscated code is crucial for tasks like malware analysis and understanding protected software, and researchers have begun evaluating LLMs for this purpose. Systematic studies assessing models like the GPT and LLaMA families find that while larger models perform well on non-obfuscated code, their analysis accuracy significantly decreases when dealing with obfuscated code, and they exhibit limited capability in generating de-obfuscated code [77].

3.3 Application of LLMs in Information and Content Security

This section explores the application of LLMs in the field of information and content security. The tasks it includes phishing and scam, harmful contents, steganography, access control, and forensics.

Phishing and scam detection. Network deception is a deliberate act of introducing false or misleading content into a network system, threatening the personal privacy and property security of users. Emails, short message service (SMS), and web advertisements are leveraged by attackers to entice users and steer them towards phishing sites, enticing them to click on malicious links [275]. LLMs can generate deceptive or false information on a large scale under specific prompts [243], making them useful for automated phishing email generation[108, 247], but compared to manual design methods, phishing emails generated by LLMs have lower click-through rates [108]. LLMs can achieve phishing email detection through prompts based on website information [145] or fine-tuning for specific email features [204, 247]. Spam emails often contain a large number of phishing emails. Labonne et al.'s research [148] has demonstrated the effectiveness of LLMs in spam email detection, showing significant advantages over traditional machine learning methods. An interesting study [34] suggests that LLMs can mimic real human interactions with scammers in an

automated and meaningless manner, thereby wasting scammers' time and resources and alleviating the nuisance of scam emails.

Harmful contents detection. Social media platforms frequently face criticism for amplifying political polarization and deteriorating public discourse. Users often contribute harmful content that reflects their political beliefs, thereby intensifying contentious and toxic discussions or participating in harmful behavior [308]. The application of LLMs in detecting harmful content can be divided into three aspects: detection of extreme political stances [103, 198], tracking of criminal activity discourse [116], and identification of social media bots [33, 280], as well as assisting human moderators in rating content against platform policies. For challenging multimodal content like memes, approaches such as EXPLAINHM leverage LLMs to generate explanatory debates between opposing personas to aid in detecting implicit harmfulness [168]. LLMs tend to express attitudes consistent with the values encoded in the programming when faced with political discourse, indicating the complexity and limitations of LLMs in handling social topics [105]. Hartvigsen et al. [194] generated a large-scale dataset of harmful and benign discourse targeting 13 minority groups using LLMs. Through validation, it was found that human annotators struggled to distinguish between LLM-generated and human-written discourse, advancing efforts in filtering and combating harmful contents.

Steganography. Steganography, as discussed in Anderson's work [15], focuses on embedding confidential data within ordinary information carriers without alerting third parties, thereby safeguarding the secrecy and security of the concealed information. Wang et al. [296] introduced a method for language steganalysis using LLMs based on few-shot learning principles, aiming to overcome the limited availability of labeled data by incorporating a small set of labeled samples along with auxiliary unlabeled samples to improve the efficiency of language steganalysis. This approach significantly improves the detection capability of existing methods in scenarios with few samples. Bauer et al. [20] used the GPT-2 model to encode ciphertext into natural language cover texts, allowing users to control the observable format of the ciphertext for covert information transmission on public platforms.

Access control. Access control aims to restrict the actions or operations permissible for a legitimate user of a computer system [251], with passwords serving as the fundamental component for its implementation. Despite the proliferation of alternative technologies, passwords continue to dominate as the preferred authentication mechanism [225]. PassGPT, a password generation model leveraging LLMs, introduces guided password generation, wherein PassGPT's sampling process generates passwords adhering to user-defined constraints. This approach outperforms existing methods utilizing Generative Adversarial Networks (GANs) by producing a larger set of previously unseen passwords, thereby demonstrating the effectiveness of LLMs in improving existing password strength estimators [244].

Forensics. In the realm of digital forensics, the successful prosecution of cybercriminals involving a wide array of digital devices hinges upon its pivotal role. The evidence retrieved through digital forensic investigations must be admissible in a court of law [257]. Scanlon and colleagues [255] delved into the potential application of LLMs within the field of digital forensics. Their exploration encompassed an assessment of LLM performance across various digital forensic scenarios, including file identification, evidence retrieval, and incident response. Their findings led to the conclusion that while LLMs currently lack the capability to function as standalone digital forensic tools, they can nonetheless serve as supplementary aids in select cases.

3.4 Application of LLMs in Hardware Security

Modern computing systems are built on System-on-Chip (SoC) architectures because they achieve high levels of integration by using multiple Intellectual Property (IP) cores. However, this also brings about new security challenges, as a vulnerability in one IP core could affect the security of the entire system. While software and firmware patches

can address many hardware security vulnerabilities, some vulnerabilities cannot be patched, and extensive security assurances are required during the design process [64]. This section explores the application of LLMs in the field of hardware security. The tasks it includes hardware vulnerability detection and hardware vulnerability repair.

Hardware vulnerability detection. LLMs can extract security properties from hardware development documents. Meng et al. [197] trained HS-BERT on hardware architecture documents such as RISC-V, OpenRISC, and MIPS, and identified 8 security vulnerabilities in the design of the OpenTitan SoC. Additionally, Paria et al. [224] used LLMs to identify security vulnerabilities from user-defined SoC specifications, map them to relevant CWEs, generate corresponding assertions, and take security measures by executing security policies.

Hardware vulnerability repair. LLMs have found application within the integrated System-on-Chip (SoC) security verification paradigm, showcasing potential in addressing diverse hardware-level security tasks such as vulnerability insertion, security assessment, verification, and the development of mitigation strategies [250]. By leveraging hardware vulnerability information, LLMs offer advice on vulnerability repair strategies, thereby improving the efficiency and accuracy of hardware vulnerability analysis and mitigation efforts [170]. In their study, Nair and colleagues [211] demonstrated that LLMs can generate hardware-level security vulnerabilities during hardware code generation and explored their utility in generating secure hardware code. They successfully produced secure hardware code for 10 CWEs at the hardware design level. Additionally, Tan et al. [8] curated a comprehensive corpus of hardware security vulnerabilities and evaluated the performance of LLMs in automating the repair of hardware vulnerabilities based on this corpus. Further investigation into automated hardware bug repair using LLMs has shown that effective prompt engineering, combined with rigorous functional and formal validation, can successfully fix security bugs in hardware description language code, outperforming synthesis-based tools [9].

Hardware IP protection. Beyond detecting and repairing vulnerabilities, the role of LLMs in relation to hardware IP protection is also under investigation, including potential misuse. Research such as LLMPirate explores how LLMs might be leveraged for hardware IP piracy, demonstrating automated frameworks that employ techniques like syntax translation and interactive feedback to rewrite hardware netlists for evading detection tools, thus highlighting potential threats and the need for robust defenses [95].

Security assertions generation. Assertion-Based Verification (ABV) using languages like SystemVerilog Assertions (SVA) is crucial for hardware security, but manually creating effective security assertions is challenging. Research is exploring the use of LLMs to automate this process; studies evaluating LLMs for generating security SVA indicate that while LLMs can produce syntactically correct assertions, the functional correctness and relevance heavily rely on the quality of the prompt context, such as providing related examples or explicit security property descriptions [137].

3.5 Application of LLMs in Blockchain Security

This section explores the application of LLMs in the field of blockchain security. The tasks it includes smart contract security and transaction anomaly detection.

Smart contract vulnerability detection. With the advancement of blockchain technology, smart contracts have emerged as a pivotal element in blockchain applications [361]. Despite their significance, the development of smart contracts can introduce vulnerabilities that pose potential risks such as financial losses. While LLMs offer automation for detecting vulnerabilities in smart contracts, the detection outcomes often exhibit a high rate of false positives [41, 57] and require careful consideration of formal properties [184]. Performance varies across different vulnerability types and is constrained by the contextual length of LLMs [41]. GPTLENS [120] divides the detection process of smart contract vulnerabilities into two phases: generation and discrimination. During the generation phase, diverse vulnerability

responses are generated, and in the discrimination phase, these responses are evaluated and ranked to mitigate false positives. Sun and colleagues [271] integrated LLMs and program analysis to identify logical vulnerabilities in smart contracts, breaking down logical vulnerability categories into scenarios and attributes. They utilized LLMs to match potential vulnerabilities and further integrated static confirmation to validate the findings of LLMs. To address risks from code reuse, SOChecker uses LLMs first to complete potentially insecure code snippets from Stack Overflow and then to identify vulnerabilities within the context of the completed code [43]. LLMs are also applied to detect specific complex semantic vulnerabilities, such as Ponzi schemes using zero-shot Chain-of-Thought prompting [309], or accounting errors through hybrid systems combining LLM-based semantic annotation with rule-based reasoning [341]. Related work also explores using LLMs to generate adversarial contracts for validating exploitability [314] or combining LLMs with GNNs and on-chain analysis to detect deployed malicious contracts before attacks occur [294].

Transaction anomaly detection. Due to the limitations of the search space and the significant manual analysis required, real-time intrusion detection systems for blockchain transactions remain challenging. Traditional methods primarily employ reward-based approaches, focusing on identifying and exploiting profitable transactions, or pattern-based techniques relying on custom rules to infer the intent of blockchain transactions and user address behavior [246, 311]. However, these methods may not accurately capture all anomalies. Therefore, more general and adaptable LLMs technology can be applied to effectively identify various abnormal transactions in real-time. Gai et al. [91] apply LLMs to dynamically and in real-time detect anomalies in blockchain transactions. Due to its unrestricted search space and independence from predefined rules or patterns, it enables the detection of a wider range of transaction anomalies.

3.6 Application of LLM Agents in Security

Beyond single-task applications, a significant recent trend is the development of LLM-based autonomous agents for security. These agents are sophisticated systems built around an LLM core, endowed with capabilities for autonomous reasoning, planning, memory, and the use of external tools to accomplish complex, multi-step goals [186, 324]. This paradigm shifts from using LLMs as simple query-response tools to employing them as autonomous problem-solvers that can interact with environments and other agents to tackle dynamic security challenges. A predominant architecture emerging is the Multi-Agent System (MAS), where specialized agents collaborate, debate, or operate in a hierarchy to achieve a common objective.

Network Security. In network security, agent-based frameworks are revolutionizing complex offensive tasks. For penetration testing, multi-agent systems like VulnBot [147] and hierarchical teams of agents [365] automate the entire attack lifecycle. These systems typically feature a planning agent that decomposes the goal, specialized agents that execute tasks like reconnaissance or tool use (e.g., using ‘nmap‘ or ‘Metasploit‘), and a reporting agent that synthesizes the findings. This collaborative approach has proven effective in exploiting even zero-day vulnerabilities. For protocol analysis, agents like RFCScan [358] employ a dual-agent structure, with an "indexing agent" processing RFC documents and a "detection agent" querying this indexed knowledge to find functional bugs.

Software and System Security. This domain has seen a surge in agentic systems that address the full lifecycle of a vulnerability. For vulnerability detection and exploitation, frameworks like FaultLine [214] establish an agentic workflow to automate the generation of proof-of-vulnerability exploits, moving beyond simple detection to active confirmation. For vulnerability repair, multi-agent systems like AutoPatch [258] create a pipeline of specialized agents, including a "Vulnerability Verifier" and a "Code Patcher," that work together to automatically fix insecure code generated by LLMs. In proactive defense, SCGAgent [254] functions as an autonomous agent that leverages security guidelines and generates unit tests to ensure newly written code is both functional and secure. For malware analysis, agents like

LAMD [237] emulate a human analyst's workflow by first using backward program slicing—a classic program analysis technique—to isolate security-critical code regions. This focused context is then fed into a tier-wise reasoning process, allowing the agent to analyze behavior from individual instructions to overall intent and autonomously detect malicious patterns in Android applications. Furthermore, agents are being developed for the complex task of reverse engineering and binary analysis. ReCopilot [42] is an expert agent trained specifically for binary analysis tasks like function name recovery from decompiled code, outperforming general-purpose models. For end-to-end analysis, Vul-BinLLM [124] detects vulnerabilities directly in compiled binaries by using an optimized decompiler and a function analysis queue to manage the LLM's context window, allowing it to analyze large, real-world binaries while maintaining overall context. A key insight from this research is that the most successful software security agents are not replacing classical tools but orchestrating them; the LLM acts as the intelligent "brain" while requiring the precise "eyes and ears" of static analyzers, debuggers, and program slicers to operate effectively.

Information and Content Security. To combat deception, researchers have developed innovative debate-driven multi-agent systems. In phishing detection, frameworks like PhishDebate [163] and similar debate-oriented models [164] utilize a team of agents. Each agent is assigned a specific role, such as analyzing URL structures, scrutinizing textual content for social engineering tactics, or performing screenshot analysis. These agents then debate their findings to reach a collective, more robust conclusion, significantly reducing false positives and outperforming single-agent approaches. Beyond deception detection, the capabilities of LLM agents also introduce novel challenges and applications. For access control, as agents gain autonomy, the risk of privilege escalation becomes a paramount concern. To address this, systems like Prompt Flow Integrity [143] have been proposed to enforce the principle of least privilege by identifying untrusted data sources and validating unsafe data flows.

Hardware Security. Hardware security verification has traditionally been a manual, labor-intensive process, making it a prime candidate for automation through LLM agents that can learn the complex semantics of hardware description languages. Frameworks like BugWhisperer [277] are leading examples, applying agent-like automation to identify vulnerabilities in SoC designs. It utilizes a specialized LLM, fine-tuned on a comprehensive hardware vulnerability database, to automatically detect security flaws at the Register-Transfer Level (RTL). Other research directions include RTL-Breaker [192], which assesses LLM security against backdoor attacks during HDL generation.

Blockchain Security. The immutable and high-value nature of blockchain systems makes smart contract auditing a critical domain where agents show immense potential. The A1 agentic system [93] exemplifies an execution-grounded approach, transforming a general LLM into an end-to-end exploit generator. The agent hypothesizes a vulnerability, tests an exploit against a forked blockchain state, and uses the deterministic execution feedback to refine its attack, allowing it to overcome hallucinations and discover complex exploits. Other advanced frameworks include Smartify [140], a multi-agent system for detecting and repairing vulnerabilities, and SmartAuditFlow [306], which uses RAG and adaptive planning to improve audit reliability. These systems show that for formal domains like blockchain, the most promising path for agents involves building tight feedback loops between the LLM's hypothesis generation and a deterministic execution harness.

RQ1 - Summary

(1) We have divided cybersecurity tasks into six domains: software and system security, network security, information and content security, hardware security, and blockchain security. We have summarized the specific applications of LLMs in these domains.
(2)We discussed 29 cybersecurity tasks and found that LLMs are most widely applied in the field of software and system security, with 119 papers covering 11 tasks.
(3) A significant emerging trend identified is the use of LLM-based autonomous agents. Our analysis highlights that these agents represent a paradigm shift, moving from performing isolated security tasks to orchestrating complex, multi-step workflows across all major security domains.

4 RQ2: WHAT LLMS HAVE BEEN EMPLOYED TO SUPPORT CYBERSECURITY TASKS?

4.1 Architecture of LLMs in Cybersecurity

Pre-trained Language Models (PLMs) have exhibited impressive capabilities across various NLP tasks [146, 199, 260, 303, 356]. Researchers have noted substantial improvements in their performance as model size increases, with surpassing certain parameter thresholds leading to significant performance gains [111, 260]. The term "Large Language Model" distinguishes language models based on the size of their parameters, specifically referring to large-sized PLMs [199, 356].However, there is no formal consensus in the academic community regarding the minimum parameter size for LLMs, as model capacity is intricately linked to training data size and overall computational resources [139]. In this study, we adopt to the LLM categorization framework introduced by Panet et al. [223], which classifies the predominant LLMs explored in our research into three architectural categories: encoder-only, encoder-decoder, and decoder-only. We also considered whether the related models are open-source. Open-source models offer higher flexibility and can acquire new knowledge through fine-tuning on specific tasks based on pre-trained models, while closed-source models can be directly called via APIs, reducing hardware expenses. This taxonomy and relevant models are shown in Table 5. We analyzed the distribution of different LLM architectures applied in various cybersecurity domains, as shown in Fig 5.

Encoder-only LLMs. Encoder-only LLMs, exemplified by BERT [65] (mentioned in 35 papers in this study) and its variants [5, 83, 100, 107, 183, 190, 252], primarily focus on understanding and encoding input information rather than generation. In cybersecurity, researchers employ these models primarily to generate contextualized embeddings for relevant data, such as source code, network traffic, or system logs, mapping complex data types into vector representations suitable for downstream tasks [3, 302].

Various prominent models, including CodeBERT [83], GraphCodeBERT [100], RoBERTa [183], CharBERT [190], DeBERTa [107], and DistilBERT [252], have gained widespread usage due to their ability to effectively process and analyze code and other security-relevant data. CodeBERT [83], for instance, is a bimodal model utilizing both natural language and source code inputs, making it suitable for tasks requiring joint understanding, such as vulnerability detection based on code and descriptions. Note that most of these aforementioned BERT variants were not initially designed for cybersecurity tasks; instead, their application in the cybersecurity field stems from their capabilities as general models adapted for code semantics interpretation and understanding. In contrast, SecureBERT [5] is a BERT variant specifically designed for cyber threat analysis tasks, and other domain-specific models like HS-BERT [197] have

Table 5. The classification of the LLMs used in the collected papers, with the number following the model indicating the count of papers that utilized that particular LLM.

	Model	Release Time	Open Source
Encoder-Only	BERT (11)	2018.10	Yes
	RoBERTa (14)	2019.07	Yes
	DistilBERT (3)	2019.10	Yes
	CodeBERT (12)	2020.02	Yes
	DeBERTa (1)	2020.06	Yes
	GraphCodeBERT (4)	2020.09	Yes
	CharBERT (1)	2020.11	Yes
Encoder-Decoder	T5 (4)	2019.10	Yes
	BART (1)	2019.10	Yes
	PLBART (4)	2021.03	Yes
	CodeT5 (14)	2021.09	Yes
	UniXcoder (2)	2022.03	Yes
	Flan-T5 (2)	2022.10	Yes
Decoder-Only	GPT-2 (9)	2019.02	Yes
	GPT-3 (4)	2020.04	Yes
	GPT-Neo (1)	2021.03	Yes
	CodeX (14)	2021.07	No
	CodeGen (6)	2022.03	Yes
	InCoder (1)	2022.04	Yes
	PaLM (3)	2022.04	No
	Jurassic-1 (1)	2022.04	No
	GPT-3.5 (78)	2022.11	No
	LLaMa (8)	2023.02	Yes
	GPT-4 (64)	2023.03	No
	Bard (13)	2023.03	No
	Claude (5)	2023.03	No
	StarCoder (8)	2023.05	Yes
	Falcon (2)	2023.06	Yes
	CodeLLaMa (10)	2023.08	Yes
	Mixtral (2)	2023.12	Yes
	CodeGemma (1)	2024.09	Yes

been developed for hardware security property extraction from documentation, demonstrating the adaptation of this architecture for diverse, specialized security needs.

Regarding the model applicability, as shown in the Figure 5, encoder-only models initially garnered attention in the fields of network security [13] and software and systems security [153, 318]. Specific applications include failure detection in large-scale system logs using BERT fine-tuned on normal behavior [129], and in 2023, this concept was extended to the field of information and content security, utilizing encoder-only models like DistilBERT and RoBERTa for phishing and spam email detection [128] and others to detect harmful content on social media platforms [33, 103, 198].

Encoder-decoder LLMs. Encoder-decoder models, such as BART [156], T5 [241], and CodeT5 [300], structure inputs and outputs, making them suitable for sequence-to-sequence cybersecurity tasks [288]. Specific variants like CodeT5 [300] and PLBART [10] utilize bimodal inputs (programming language and text), enhancing code comprehension

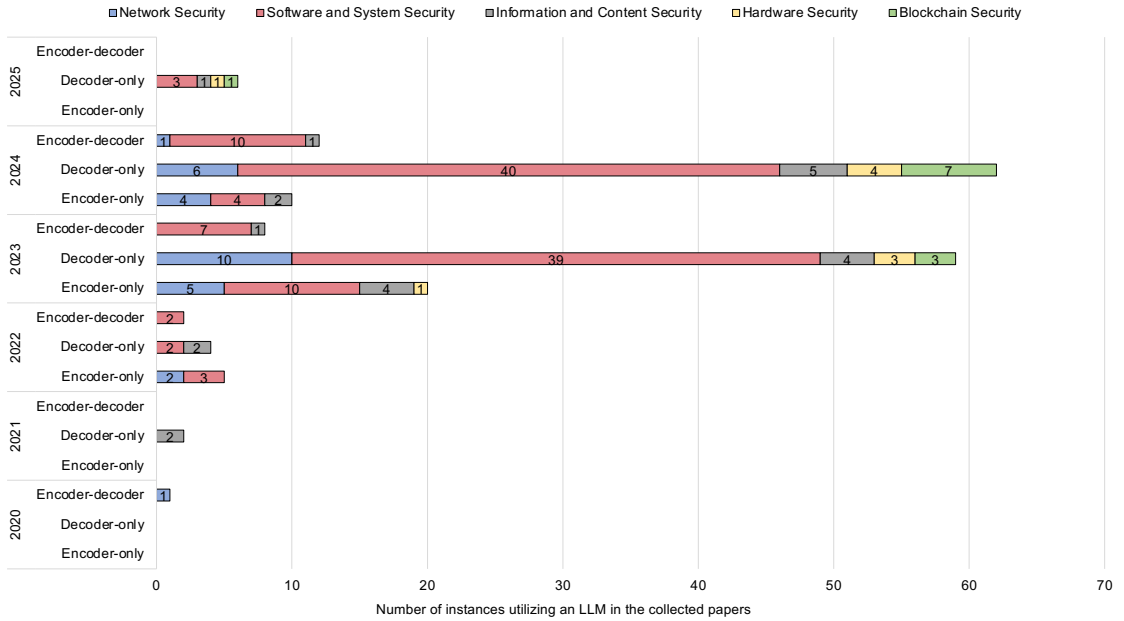


Fig. 5. Distribution and trend of different model architectures.

for security analysis [312]. This architecture has demonstrated effectiveness in various security applications, including vulnerability repair, where T5-based models like VulRepair [90] achieve strong performance, although effectiveness can vary depending on the language, as noted in studies on Java vulnerability repair using models like CodeT5 and PLBART [312]. Encoder-decoder models like CodeT5 are also utilized within larger frameworks for tasks such as vulnerability detection, for instance, by generating code comments for structural analysis in SCALE [307]. Furthermore, models like Flan-T5 have been specifically fine-tuned for spam and phishing email detection [128, 148]. Other applications include program fuzzing [63] and assisting reverse engineering [17].

Decoder-only LLMs. Decoder-only LLMs utilize an autoregressive approach, generating outputs token-by-token, making them well-suited for tasks requiring detailed generation [239], such as security analyses [132], advisories [77], and code generation [212].

Prominent examples include the GPT series (GPT-2 [240], GPT-3 [30], GPT-3.5 [217], GPT-4 [219]) and specialized versions like Codex [44]. Among these, GPT-3.5 and GPT-4 are the models most frequently used to address various cybersecurity issues in this study, covering almost all cybersecurity applications [61, 80, 85, 255]. Their strong few-shot learning abilities allow rapid development of new cybersecurity capabilities with minimal fine-tuning. Open-source models like GPT-Neo [24], LLaMa [283], and Falcon [230] also follow this architecture. Additionally, decoder-only code generation LLMs such as CodeGen [213], InCoder [88], StarCoder [161], Codex [44] and CodeLLaMa [248] have been widely used for bug detection and repair, as well as for vulnerability repair [312, 320, 328]. The large context window characteristic of decoder-only models allows them to take in and utilize more context relevant to the cybersecurity task, like related vulnerabilities, reports, and code snippets.

Due to the powerful natural language generation capabilities of the decoder-only architecture, researchers initially attempted to apply it to the generation of fake cyber threat intelligence [243]. Decoder-only LLMs have gained prominence in recent years, especially in 2023 and 2024 as shown in Figure 5, witnessing a surge in development and commercial adoption by leading Internet companies. For instance, Google introduced Bard [96], while Meta unveiled LLaMa [283]. While GPT-4 and its derivative application, ChatGPT, quickly found integration into various cybersecurity tasks, these other newer models have yet to see similarly widespread adoption in the cybersecurity domain reported in the literature surveyed.

4.2 Trend Analysis

Illustrated in Figure 5, from 2020 to 2025 (data available up to early 2025), there have been significant shifts in the preference and utilization of LLM architectures across cybersecurity tasks. The selection of decoder-only, encoder-decoder, and encoder-only structures has influenced diverse research directions and solutions in the cybersecurity field. This examination delves into the trends regarding the adoption of these architectures over time, reflecting the evolving landscape of LLM applications for cybersecurity tasks.

Table 6. Overview of the distribution of LLMs in the open-source community.

(a) Top 20 most downloaded models on Huggingface.

Model	Architecture
BERT-base	Encoder-only
DistilBERT-base	Encoder-only
GPT2	Decoder-only
RoBERTa-large	Encoder-only
RoBERTa-base	Encoder-only
xlm-RoBERTa-large	Encoder-only
xlm-RoBERTa-base	Encoder-only
DeBERTa-base	Encoder-only
Qwen-VL-Chat	Decoder-only
T5-small	Decoder-encoder
BERT-base-cased	Encoder-only
T5-base	Decoder-encoder
BERT-base-uncased	Encoder-only
CamemBERT-base	Encoder-only
DistilGPT2	Decoder-only
DistilRoBERTa-base	Encoder-only
LLaMa3-8B	Decoder-only
ALBERT-base-v2	Encoder-only
DeBERTa-v3-base	Encoder-only
ByT5-small	Decoder-encoder

(b) Top 20 most liked models on Huggingface.

Model	Architecture
BLOOM-176B	Decoder-only
LLaMa3-8B	Decoder-only
LLaMa2-7B	Decoder-only
Mixtral-8x7B	Decoder-only
Mixtral-7B	Decoder-only
Phi-2	Decoder-encoder
Gemma-7B	Decoder-only
ChatGLM-6B	Decoder-only
StarCoder	Decoder-only
Falcon-40B	Decoder-only
Grok-1	Decoder-only
ChatGLM2-6B	Decoder-only
GPT2	Decoder-only
Dolly-v2-12B	Decoder-only
BERT-base	Encoder-only
Zephyr-7B	Decoder-only
OpenELM	Decoder-only
Phi-1.5	Decoder-encoder
Yi-34B	Decoder-only
Flan-T5	Decoder-encoder

In 2020 and 2021, the use of LLMs in cybersecurity was limited, with only 1 and 2 instances identified respectively. In 2020, encoder-decoder LLMs were the sole architecture used (1 instance). However, in 2021, the focus shifted exclusively to decoder-only LLMs (2 instances). This early adoption pattern may be attributed to the research emphasis on LLM performance in natural language processing tasks and innovations in LLM architectures during this period [30, 139].

The year 2022 marked a significant turning point, with the number of instances employing LLMs for cybersecurity tasks surging to 11, surpassing the combined total from the previous two years. This year also saw increased diversity in the LLM architectures used. Encoder-only LLMs, valued for their representation learning and classification abilities, were utilized in 5 instances (45%). Encoder-decoder LLMs, with their strong performance on well-defined tasks, were featured in 2 instances (18%), while decoder-only LLMs, leveraging their knowledge recall and few-shot learning capabilities, garnered 4 instances (36%). This varied distribution highlights the active exploration of different architectures to address the diverse needs and challenges in cybersecurity.

The years 2023 and 2024 witnessed a dramatic surge in research and a clear shift towards decoder-only LLMs becoming the predominant architecture. This trend is closely tied to the powerful text comprehension, reasoning capabilities [68, 131, 191, 222, 304], and open-ended generation demonstrated by chatbots like ChatGPT. These decoder-only models often require minimal fine-tuning and can generate both syntactically correct and functionally relevant code snippets [149, 249]. In 2023 (total 86 instances), decoder-only models constituted approximately 69% (59 instances), while encoder-only usage peaked at 23% (20 instances) and encoder-decoder models represented 8% (7 instances). This dominance of decoder-only models continued strongly into 2024 (total 85 instances), accounting for roughly 73% (62 instances). Importantly, unlike the suggestion of complete replacement, both encoder-decoder (12 instances, 14%) and encoder-only (11 instances, 13%) architectures continued to be utilized in 2024 across various domains, indicating their ongoing relevance for specific tasks. The limited data for early 2025 (6 instances) exclusively features decoder-only models, suggesting their continued prominence.

The dominance of decoder-only LLMs in cybersecurity research aligns with the broader trends in the LLM community. An analysis of the top 20 most liked and downloaded LLMs on Huggingface [1], illustrated in Table 6), a popular open-source model community, reveals a divergence: encoder-only models like BERT and its variants represent the clear majority of the most downloaded models (13 out of 20), whereas decoder-only models overwhelmingly dominate the most liked list (16 out of 20). This indicates a strong community preference and excitement for the potential of decoder-only models to handle complex, open-ended tasks. The growing interest in decoder-only LLMs can be attributed to their strong generation, knowledge, and few-shot learning abilities, which make them well-suited for the diverse challenges in cybersecurity. However, the larger parameter size of these models compared to encoder-only models may limit their current adoption due to the scarcity of computational resources [81].

In summary, while the cybersecurity field leverages all three main LLM architectures, recent years (2023-2024) have shown a clear dominance of decoder-only models, likely driven by their generative strengths and few-shot capabilities popularized by models like ChatGPT. However, the continued application of encoder-only and encoder-decoder models underscores their specific advantages for certain tasks, indicating a diverse rather than monolithic architectural landscape in LLM4Security.

RQ2 - Summary

(1) We have gathered papers utilizing over 30 distinct LLMs for cybersecurity tasks. These LLMs have been categorized into three groups based on their underlying architecture or principles: encoder-only, encoder-decoder, and decoder-only LLMs.
(2) We analyzed the trend in employing LLMs for cybersecurity tasks, revealing that decoder-only architectures are the most prevalent. Specifically, over 15 LLMs fall into the decoder-only category, and a total of 133 papers have investigated the utilization of decoder-only LLMs in cybersecurity tasks.
(3) Our analysis reveals a clear trend of **task-architecture alignment**. While decoder-only models have become dominant since 2023 for their generative and reasoning capabilities, encoder-only and encoder-decoder architectures remain highly relevant for specific niches. Encoder-only models are consistently chosen for representation-heavy classification tasks, while encoder-decoder models are often preferred for structured sequence-to-sequence tasks like program repair.

5 RQ3: WHAT DOMAIN SPECIFICATION TECHNIQUES ARE USED TO ADAPT LLMS TO SECURITY TASKS?

LLMs have demonstrated their efficacy across various intelligent tasks [136]. Initially, these models undergo pre-training on extensive unlabeled corpora, followed by fine-tuning for downstream tasks. However, discrepancies in input formats between pre-training and downstream tasks pose challenges in leveraging the knowledge encoded within LLMs efficiently. The techniques employed with LLMs for security tasks can be broadly classified into three categories: prompt engineering, fine-tuning, and external augmentation. We will delve into a comprehensive analysis of these three categories and further explore their subtypes, as well as summarize the connections between LLM techniques and various security tasks.

5.1 Fine-tuning LLMs for Security Tasks

Fine-tuning techniques are extensively utilized across various downstream tasks in NLP [269], representing a crucial step in adapting pre-trained LLMs, whose initial training might not sufficiently cover the nuances of specialized domains like cybersecurity. This process involves further training the model on task-relevant datasets to adjust its parameters, aligning its capabilities more closely with specific security objectives [67, 236]. The extent and nature of fine-tuning often depend upon task complexity and dataset size [67, 236]. Importantly, fine-tuning can mitigate the constraints posed by model size, enabling smaller models fine-tuned for specific tasks to potentially outperform larger, general-purpose models lacking task-specific adaptation [142, 359]. In our surveyed literature (185 papers), fine-tuning was employed in 57 studies (31%). Its application varies significantly across domains: it was most prevalent in information and content security (10 out of 20 papers, 50%) and software and system security (43 out of 119 papers, 36%), with moderate use in network security (4 out of 26 papers, 15%). Notably, no papers applying fine-tuning were identified in the hardware security or blockchain security domains within our selection.

The primary motivation for fine-tuning in LLM4Security is to imbue general-purpose LLMs with domain-specific knowledge and capabilities, tailoring them for specific security tasks. For example, in software vulnerability management, models are fine-tuned to perform classification or generation. RealVul fine-tunes models like CodeBERT and CodeT5 on augmented datasets of PHP code snippets to improve vulnerability classification [35]. VulLLM employs multi-task

instruction fine-tuning on models including CodeBERT and CodeT5, training them simultaneously for vulnerability detection, localization, and explanation to enhance generalization [69]. For vulnerability repair, fine-tuning on datasets containing vulnerable code and patches allows models like T5 [112] or other code LLMs [121, 299, 312] to learn repair patterns. VulAdvisor further adapts this by fine-tuning CodeT5 with context-aware attention to generate natural language repair suggestions instead of direct code patches [346]. Similarly, in information security, fine-tuning is applied to tasks like harmful content detection; EXPLAINHM fine-tunes a Flan-T5 model to classify harmful memes based on debates generated by a larger multimodal LLM [168]. Fine-tuning is also used for adapting models to specific data types like network traffic [85] or system logs [141] for anomaly detection. This adaptation process typically involves preparing a high-quality dataset representative of the target task and using established training frameworks [359] to update the model's parameters.

Different strategies exist regarding the scope of parameter updates during fine-tuning. Full fine-tuning adjusts all parameters of the LLM, offering the potential for high adaptability and performance, particularly when the target task differs significantly from the pre-training objectives [188]. This approach is often employed when feasible, for instance, in tasks like bug repair [121, 226, 299, 338] or vulnerability repair [90, 312, 347] using dedicated datasets. However, full fine-tuning demands significant computational resources. Alternatively, Parameter-Efficient Fine-Tuning (PEFT) methods update only a small subset of parameters or add trainable adapters, significantly reducing computational and memory costs [262]. PEFT techniques observed in LLM4Security include adapter-tuning [114, 153, 328], prompt-tuning [47, 155], and Low-Rank Adaptation (LoRA) [117, 263], alongside API-based fine-tuning for closed-source models [12, 63, 116, 133, 142, 218]. These PEFT methods make fine-tuning more accessible, especially for very large models or in resource-constrained environments, while still achieving effective task adaptation.

Beyond standard supervised fine-tuning, Reinforcement Learning (RL) based strategies are gaining attention for aligning LLMs more closely with specific security objectives or leveraging their outputs to guide automated processes [59, 86, 222]. These approaches often involve optimizing an agent's policy using reward signals derived from task execution or human feedback, sometimes incorporating LLM-generated insights into the reward function [274]. For instance, in web application security, RL can fine-tune an LLM to generate more effective web fuzzing inputs by rewarding payload sequences that successfully bypass defenses or trigger desired responses [166]. Similarly, RL techniques can enhance log analysis by optimizing models based on the accuracy or utility of the identified anomalies [101]. In the domain of fuzzing complex systems like JavaScript engines, CovRL utilizes coverage-guided RL: it processes execution coverage using methods like TF-IDF to assign higher importance to rarely executed paths, then uses this weighted coverage to generate a reward signal, thereby fine-tuning an LLM mutator to produce mutations more likely to explore these less-covered, potentially bug-rich areas [74]. For automated program repair, RePair leverages RL with process-based feedback; instead of just learning from fixed code examples, the LLM's policy for generating patches is directly rewarded based on the outcome of executing the generated code against test suites, thus encouraging the model to learn repair strategies that lead to functional correctness [357]. In a different application connecting LLMs and RL, CrashTranslator utilizes LLMs first to analyze stack traces and compute semantic similarities with UI elements; this information then shapes the reward function for an RL agent that learns an optimized UI exploration policy to effectively reproduce mobile application crashes [123].

Despite its benefits, fine-tuning LLMs for security tasks faces persistent challenges. Creating high-quality, large-scale, and representative labeled datasets remains a primary bottleneck; cybersecurity data is often scarce, imbalanced, noisy, or may contain inherent biases reflecting only known threats or specific environments, which can limit the model's real-world applicability [67]. Furthermore, while generally less computationally intensive than pre-training, fine-tuning

still demands considerable resources, particularly for full parameter updates on large models [102]. Performance limitations also exist. There is a significant risk of overfitting to the specific patterns present in the fine-tuning dataset, which can impair the model's ability to generalize to novel or unseen vulnerability types, attack vectors, or code constructs encountered in practice [203]. Relatedly, catastrophic forgetting can occur, where the model loses some of its general language or reasoning capabilities acquired during pre-training [187]. A critical concern, especially relevant to security, is that fine-tuning can inadvertently compromise safety alignments embedded during pre-training, potentially making models more susceptible to generating insecure code or harmful content if not carefully managed [71, 236].

5.2 Prompting LLMs for Security Tasks

Recent studies in natural language processing highlight the significance of prompt engineering [177] as an emerging fine-tuning approach aimed at bridging the gap between the output expectations of LLMs during pretraining and downstream tasks. This strategy has demonstrated notable success across various NLP applications. Incorporating meticulously crafted prompts as features in prompt engineering has emerged as a fundamental technique for enriching interactions with LLMs like ChatGPT, Bard, among others. These customized prompts serve a dual purpose: they direct the LLMs towards generating specific outputs while also serving as an interface for tapping into the vast knowledge encapsulated within these models.

In prompt engineering, utilizing inserted prompts to provide task-specific knowledge is especially beneficial for security tasks with limited data features. This becomes crucial when conventional datasets (such as network threat reports, harmful content on social media, code vulnerability datasets, etc.) are restricted or do not offer the level of detail needed for particular security tasks. For example, in handling cyber threat analysis tasks [264], one can construct prompts by incorporating the current state of the network security posture. This prompts LLMs to learn directly from the flow features in a zero-shot learning manner [321], extracting structured network threat intelligence from unstructured data, providing standardized threat descriptions, and formalized categorization. In the context of program fuzzing tasks [118], multiple individual test cases can be integrated into a prompt, assisting LLMs in learning the features of test cases and generating new ones through few-shot learning [21], even with limited input. For tasks such as penetration testing [61] and hardware vulnerability verification [250], which involve multiple steps and strict logical reasoning relationships between steps, one can utilize a chain of thought (COT) [304] to guide the customization of prompts. This assists LLMs in process reasoning and guides them to autonomously complete tasks step by step.

In LLM4Security, almost all security tasks listed in Table4 involve prompt engineering, highlighting the indispensable role of prompts. In conclusion, recent research emphasizes the crucial role of prompt engineering in enhancing the performance of LLMs for targeted security tasks, thereby aiding in the development of automated security task solutions.

5.3 External Augmentation

While LLMs undergo thorough pre-training on extensive datasets, employing them directly for tackling complex tasks in security domains faces numerous challenges due to the diversity of domain data, the complexity of domain expertise, and the specificity of domain goals [171]. Several studies in LLM4Security introduce external augmentation methods to enhance the application of LLMs in addressing security issues. These external augmentation techniques facilitate improved interaction with LLMs, bridging gaps in their knowledge base, and maximizing their capability to produce dependable outputs based on their existing knowledge.

We summarized the external augmentation techniques combined with LLMs in previous studies, as shown in Table 7, identifying 8 distinct categories. The first augmentation technique is feature augmentation. The effectiveness of LLMs

Table 7. External augmentation techniques involved in prior studies.

Augmentation technique	Description	Examples	Reference
Features augmentation	Incorporating task-relevant features implicitly present in the dataset into prompts.	Adding bug descriptions, bug locations, code context or resampling for imbalanced traffic.	[133] [343] [316] [128] [12] [349] [176]
External retrieval	Retrieving task-relevant information available in external knowledge bases as input.	An external structured corpus of network threat intelligence,a hybrid patch retriever for fix pattern mining.	[70] [299] [231] [297] [313] [353] [130] [259] [336] [184]
External tools	Analysis results from specialized tools serving as auxiliary inputs or validate the result with external tools.	Static analysis tools, penetration testing tools, Rule-Based Reasoning Engine	[104] [14] [17] [35] [212] [327] [50] [353] [346] [123] [341] [8]
Task-adaptive training	Different training strategies from pre-training to enhance the model's adaptability to the task.	Contrastive learning, transfer learning, distillation.	[78] [153] [347] [229] [101] [166] [276] [33]
Inter-model interaction	ntroducing multiple models (which can be LLMs or other models) to collaborate and interact.	Multiple LLMs collaboration, graph neural networks, gGAN	[33] [330] [276] [212] [123] [209] [294]
Rebroadcasting	Applicable to multi-step tasks, broad-casting the output results of each step iteratively as part of the prompt for the next step.	Difficulty-based patch example replay, variables' name propagation	[326] [338] [327]
Post-process	Customizing special processing strategies for LLMs' outputs to better match task requirements.	Post-processing based on Levenshtein distance to mitigate hallucinations, formal verification for generated code	[47] [281] [185] [82] [314] [294]
Workflow Integration	Integrating LLMs as components within specific task workflows, incorporating various forms of interactive feedback.	LLM-guided fuzzing strategy generation, Self-check reasoning, Coverage-guided feedback loop, Integrate LLM outputs into the human rating	[196] [297] [292] [189] [280] [168]

in handling downstream tasks heavily relies on the features included in the prompts. We have observed that many studies employing LLMs for security tasks extract contextual relationships or other implicit features from raw data and integrate them with the original data to customize prompts. These implicit features encompass descriptions of vulnerabilities [349], bug locations [133], threat flow graphs [176], and more. Incorporating these implicit features alongside raw data leads to enhanced performance compared to constructing prompts solely from raw data. The next augmentation technique is external retrieval. External knowledge repositories can mitigate the hallucinations or errors arising from the lack of domain expertise in LLMs. LLMs can continually interact with external knowledge repositories during pipeline processing and retrieve knowledge relevant to security tasks to provide superior solutions [231]. Next are external tools; rule-based external tools can serve as specialized external knowledge repositories, and in addressing security tasks, LLMs can utilize results from external tools (like static analyzers or formal methods) to rectify their outputs, thereby avoiding redundancy and errors[14, 104, 323]. The fourth augmentation technique is task-adaptive training. Existing studies adopt various training strategies (beyond standard fine-tuning discussed in Section 5.1), like contrastive learning or distillation, to strengthen LLMs' adaptability to complex security tasks, enabling them to generate more targeted outputs [46, 78, 276]. The fifth augmentation technique, inter-model interaction, has garnered significant attention when a single LLM may struggle to handle complex and intricate tasks. Decomposing the pipeline process and introducing multiple LLMs for enhanced performance have been explored [330]. This approach leverages collaboration and interaction among models to harness the underlying knowledge base advantages of each LLM. When a single interaction is insufficient to support LLMs in tasks such as variable name recovery or generating complex program patches [326, 338], rebroadcasting is used: constructing prompts for LLMs multiple times continuously where the output results of each step are broadcast iteratively as part of the prompt for the next step. This helps reinforce the contextual relationship between each interaction, thereby reducing error rates. The next augmentation technique is post-processing, where LLMs' outputs are validated or processed for certain security tasks requiring specific types of output [281]. This process helps mitigate issues such as hallucinations arising from the lack of domain knowledge in LLMs [47]. The last strategy is workflow integration, which involves integrating LLMs as components within specific task workflows, incorporating various forms of interactive feedback. This feedback, such as coverage metrics guiding fuzzing strategies [189, 292], self-check reasoning steps [292], or outputs being integrated into human rating systems [280], allows the workflow's context and objectives to steer the LLM's behavior.

External augmentation techniques have significantly boosted the effectiveness of LLMs across various security tasks, yielding competitive performance. From these studies, it is evident that external augmentation techniques have the potential to address issues such as hallucinations and high false positive rates caused by deficiencies in LLMs' domain knowledge and task alignment. We believe that the integration of LLMs with external techniques will be a trend in the development of automated security task solutions.

RQ3 - Summary

(1) We summarize the domain-specific techniques used in previous research to apply LLMs to security tasks, including prompt engineering, fine-tuning, and external augmentation.
(2) Prompt engineering is the most widely used domain technique, with almost all 185 papers employing this approach. Fine-tuning techniques were used in 31% of the papers, while task-specific external augmentation techniques were adopted in 36% of the papers.
(3) We categorize and discuss the fine-tuning and external augmentation techniques mentioned in these papers, and analyze their relevance to specific security tasks.
(4) A key insight is that state-of-the-art LLM4Security applications are increasingly designed as hybrid systems. While prompt engineering is universal, the most sophisticated approaches specialize the model's capabilities, either through internal adaptation (fine-tuning) or by integrating it with external data and tools (external augmentation). This indicates a clear trend towards using LLMs as a core reasoning engine within a larger, more reliable security workflow, rather than as standalone oracles.

6 RQ4: WHAT IS THE DIFFERENCE IN DATA COLLECTION AND PRE-PROCESSING WHEN APPLYING LLMS TO SECURITY TASKS?

Data plays a vital role throughout the model training process [339]. Initially, collecting diverse and rich data is crucial to enable the model to handle a wide range of scenarios and contexts effectively. Following this, categorizing the data helps specify the model's training objectives and avoid ambiguity and misinterpretation. Additionally, preprocessing the data is essential to clean and refine it, thereby enhancing its quality. In this chapter, we examine the methods of data collection, categorization, and preprocessing as described in the literature.

6.1 Data Collection

Data plays an indispensable and pivotal role in the training of LLMs, influencing the model's capacity for generalization, effectiveness, and performance [272]. Sufficient, high-quality, and diverse data are imperative to facilitate the model's comprehensive understanding of task characteristics and patterns, optimize parameters, and ensure the reliability of validation and testing. Initially, we explore the techniques employed for dataset acquisition. Through an examination of data collection methods, we classify data sources into four categories: open-source datasets, collected datasets, constructed datasets, and industrial datasets.

Open-source datasets. Open-source datasets refer to datasets that are publicly accessible and distributed through open-source platforms or online repositories [33, 41, 193, 342]. For example, the UNSW-NB15 dataset contains 175,341 network connection records, including summary information, network connection features, and traffic statistics. The network connections in the dataset are labeled as normal traffic or one of nine different types of attacks [208]. These are commonly used for benchmarking models or as a basis for fine-tuning on established tasks. The credibility of these datasets is ensured by their open-source nature, which also allows for community-driven updates. This makes them dependable resources for academic research.

Collected datasets. Researchers gather collected datasets directly from various sources, such as major websites, forums, blogs, and social media platforms. These datasets may include comments from GitHub, harmful content from

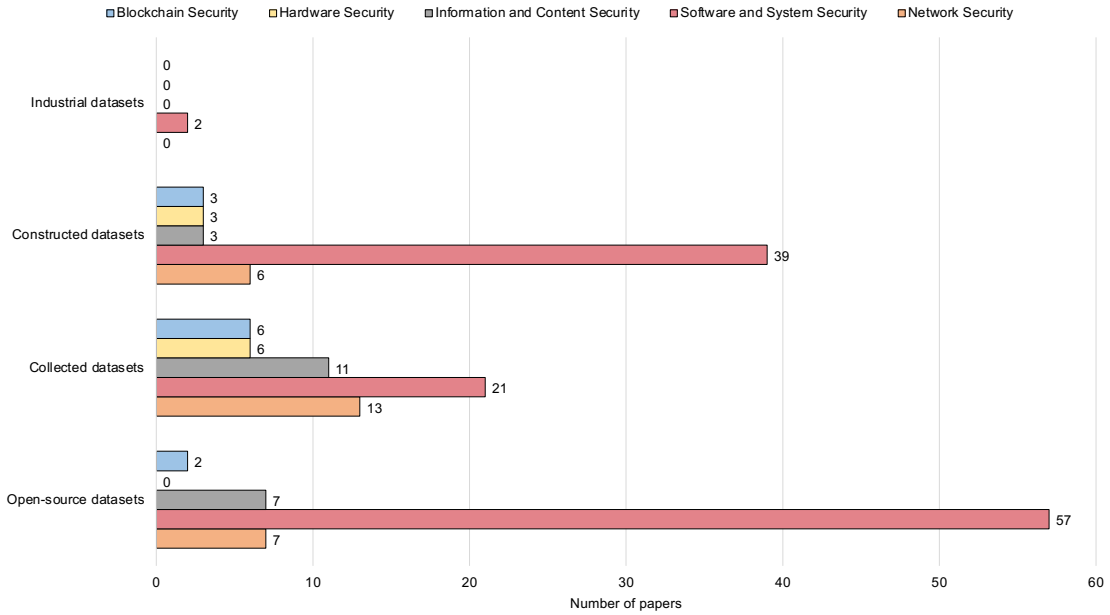


Fig. 6. The collection strategies of datasets in LLM4Security.

social media, or vulnerability information from CVE websites, tailored to specific research questions. Such data is often curated specifically for training or evaluating models on niche or newly defined security problems.

Constructed datasets. The constructed dataset refers to a specialized dataset created by researchers through the modification or augmentation of existing datasets to better suit their specific research goals [8, 145, 205, 312]. These changes could be made through manual or semi-automated processes, which might entail creating test sets tailored to specific domains, annotating datasets, or generating synthetic data. Generated synthetic data, for instance, often serves to augment training sets for fine-tuning LLMs when real-world data is scarce (as discussed further in Section 5). For instance, researchers might gather information on web vulnerabilities and the corresponding penetration testing methods, structure them into predefined templates to form vulnerability scenarios, which can then be used for evaluating LLM reasoning capabilities via prompting [61].

Industrial datasets. Industrial datasets are data obtained from real-world commercial or industrial settings, typically consisting of industrial applications, user behavior logs, and other sensitive information [174, 343]. These datasets are particularly valuable for research aimed at training and evaluating models intended for real-world deployment and application scenarios.

The Figure 6 illustrates the data collection strategies for LLM-related datasets. From the data depicted in the Figure 6, it can be observed that 73 studies utilize open-source datasets. The utilization of open-source datasets is predominantly attributed to their authenticity and credibility. These datasets are typically comprised of real-world data sourced from diverse origins, including previous related research, thereby ensuring a high degree of reliability and stability to real-world scenarios. This authenticity enables LLMs to learn from genuine examples, facilitating a deeper understanding of real-world security tasks and ultimately improving their performance when used for fine-tuning or evaluation.

Additionally, due to the recent emergence of LLMs, there is indeed a challenge of the lack of suitable training sets. Hence, researchers often collect data from websites or social media (collected datasets) and construct datasets to serve as specific training, fine-tuning, or evaluation sets to make the data more suitable for specific security tasks. We also analyzed the relationship between data collection strategies and the security domain. In certain domains such as network security, the preference for collecting datasets surpasses that of using open-source datasets. This indicates that obtaining data for applying LLMs to certain security tasks is still inconvenient. Among the 185 papers examined, only 2 studies utilized industrial datasets. This indicates a potential gap between the characteristics of datasets used in academic research and those in real-world industrial settings. This difference underscores the importance of future research exploring industrial datasets to ensure the applicability and robustness of LLMs across academic and industrial domains. Some papers focus on exploring the use of existing LLMs, such as ChatGPT, in security tasks [210, 211]. These papers often do not specify the datasets used for model training, as LLMs like ChatGPT typically do not require users to prepare their own training data for application scenarios but are instead guided via prompting or in-context learning.

6.2 Types of Datasets

The choice of data types plays a crucial role in shaping the architecture and selection of LLMs, as they directly influence the extraction of implicit features and subsequent decision-making by the model during both training and inference. This decision significantly impacts the overall performance and ability of LLMs to generalize [181]. We conducted a thorough analysis and categorization of the data types utilized in LLM4Security research. Through examining the interplay between data types, model architectures, and task demands, our goal is to highlight the vital significance of data types in effectively applying LLMs to security-related tasks.

Data type categorization. We categorize all datasets into three types: code-based, text-based, and combined data types. Table 8 provides a detailed breakdown of the specific data included in each category, derived from 185 studies. The analysis reveals that the majority of instances rely on code-based datasets, constituting a total of 90 instances. This dominance underscores the inherent code analysis capabilities of LLMs when applied for security tasks. These models demonstrate proficiency in understanding and processing code data, making code-based datasets suitable for fine-tuning models for tasks like vulnerability detection, program fuzzing, and bug repair. Their capacity to handle and learn from extensive code data enables LLMs to offer robust insights and solutions for various security applications either through fine-tuning or direct prompting.

Text-based datasets (total 60 instances) encompass a wide variety of textual information relevant to cybersecurity. While understanding the intricacies of training data might not be crucial for closed-source LLMs like ChatGPT, insights into data handling techniques for other models are still valuable. These datasets range from structured formats like prompts used for direct task execution [61, 176] or logs used for fine-tuning anomaly detection models [51, 141, 174], to unstructured natural language found in bug reports [70, 78], CVE reports [3, 5], commit messages [313, 353], security policies [212], hardware documentation [197], and social media content [103, 116]. Specific security tasks necessitate particular text data inputs, often utilized for fine-tuning specialized classifiers (e.g., for harmful content) or providing context for LLM analysis via prompting (e.g., CVE reports).

The prevalence of vulnerable code (34 instances), source code (23), and bug-fix pairs (16) in code-based datasets can be attributed to their ability to effectively meet task requirements for fine-tuning or evaluation. Vulnerable code naturally exhibits semantic features of code containing vulnerabilities to LLMs, thereby highlighting the distinguishing traits of vulnerable code when juxtaposed with normal code snippets when training detection models. A similar rationale applies to bug-fix pairs, which are essential for fine-tuning program repair models. Source code serves as the backbone of any

Table 8. Data types of datasets involved in prior studies.

Category	Data type	Studies	Total	References
Code-based datasets	Vulnerable code	24	90	[49] [84] [47] [54] [180] [279] [342] [354] [142] [284] [176] [312] [38] [14] [7] [41] [285] [35] [307] [50] [309] [314] [294] [341]
	Source code	23		[19] [118] [62] [330] [326] [270] [17] [229] [266] [176] [271] [57] [120] [91] [119] [77] [189] [74] [337] [261] [32] [79] [95]
	Bug-fix pairs	16		[133] [162] [351] [338] [226] [216] [318] [121] [348] [320] [319] [299] [349] [263] [336] [173]
	Buggy code	10		[158] [153] [265] [226] [121] [63] [8] [329] [357] [290]
	Traffic packages	4		[202] [85] [193] [13]
	Patches	3		[142] [152] [276]
	Code changes	3		[328] [70] [305]
	Vulnerability-fix pairs	3		[347] [90] [346]
	Web attack payloads	2		[175] [166]
	Subject protocol programs	1		[195]
	Vulnerable programs	1		[227]
Text-based datasets	Prompts	19	60	[12] [205] [61] [278] [39] [104] [76] [282] [343] [228] [27] [108] [247] [194] [255] [250] [170] [212] [184]
	Log messages	8		[174] [51] [141] [235] [101] [259] [132] [129]
	Social media contents	7		[103] [116] [33] [198] [296] [280] [168]
	Spam messages	4		[148] [204] [34] [128]
	Bug reports	4		[78] [153] [70] [82]
	Accident reports	2		[99] [185]
	Attack descriptions	2		[29] [80]
	CVE reports	2		[3] [4]
	Commit messages	2		[351] [299]
	Cyber threat intelligence data	2		[243] [145]
	Protocol Messages	2		[196] [297]
	Program documentations	2		[316] [292]
	Top-level domains	1		[178]
	Cellular network specifications	1		[242]
	Security reports	1		[5]
	Threat reports	1		[264]
	Structured threat information	1		[231]
	Antivirus scan reports	1		[134]
	Passwords	1		[244]
	Hardware documentations	1		[197]
Combined datasets	Vulnerable code and vulnerability descriptions	7	23	[180] [47] [212] [169] [130] [159] [43]
	Buggy code and bug reports	4		[353] [123] [295] [274]
	Vulnerability-fix pairs and vulnerability descriptions	3		[325] [69] [8]
	Bug-fix pairs and bug reports	3		[53] [362] [209]
	Vulnerability-fix pairs and repair strategies	1		[151]
	Code changes and vulnerability descriptions	1		[159]
	Source code and program documentations	1		[343]
	Buggy code and commit messages	1		[112]
	Assertion code and hardware module descriptions	1		[137]
	Decompiled code and variable names	1		[327]

Table 9. The data preprocessing techniques for code-based datasets.

Preprocessing techniques	Description	Examples	References
Data extraction	Retrieve pertinent code segments from code-based datasets tailored to specific security tasks, accommodating various levels of granularity and specific task demands.	Token-level, statement-level, class-level, traffic flow.	[270] [37] [202]
Duplicated instance deletion	Eliminate duplicate instances from the dataset to maintain data integrity and avoid repetition during the training phase.	Removal of duplicate code, annotations, and obvious vulnerability indicators in function names.	[279] [342] [349]
Unqualified data deletion	Remove unfit data by implementing filtering criteria to preserve suitable samples, ensuring the dataset's quality and suitability for diverse security tasks.	Remove or anonymize comments and information that may provide obvious hints about the vulnerability (package, variable names, and strings,etc.).	[84] [142] [338] [226]
Code representation	Represent code as tokens, code graph, AST, etc.	Tokenize source or binary code as tokens or transform code into call graph.	[354] [318] [121] [341]
Data segmentation	Divide the dataset into training, validation, and testing subsets for model training, parameter tuning, and performance evaluation.	Partition the dataset based on specific criteria, which may include division into training, validation, or testing subsets.	[328] [266]

software project, encompassing the logic and instructions that define program behavior. Thus, having a substantial amount of source code data is essential for training LLMs (either during pre-training or fine-tuning) to grasp the intricacies of programs, enabling them to proficiently generate, analyze, and comprehend code across various security tasks. Additionally, commonly used data types such as buggy code (10) and traffic packages (4), are also widespread for training detection or analysis models.

Combined datasets, integrating multiple data types, represent a growing approach (23 instances identified). This strategy often pairs code artifacts with relevant natural language text, such as combining vulnerable code with vulnerability descriptions [47, 180], buggy code with bug reports [123, 353], or source code with program documentation [343]. The key advantage lies in leveraging the LLM's ability to process both structured code and unstructured text simultaneously [293]. By providing natural language context (e.g., descriptions of what a vulnerability entails or what a function should do), these combined datasets help LLMs achieve a deeper semantic understanding of the code during fine-tuning or prompting, leading to more accurate vulnerability detection, better code repair suggestions, or more effective code analysis compared to using code data alone.

6.3 Data Pre-processing

When training and using LLMs, it's important to preprocess the initial dataset to obtain clean and appropriate data for model training [153]. Data preprocessing involves tasks like cleaning, reducing noise, and normalization. Different types of data may require different preprocessing methods to improve the performance and effectiveness of LLMs in security tasks, maintaining data consistency and quality. This section will provide a detailed explanation of the data preprocessing steps customized for the two main types of datasets: those based on code and those based on text.

Table 10. The data preprocessing techniques for text-based datasets.

Preprocessing techniques	Description	Examples	References
Data extraction	Retrieve appropriate text from documentation based on various security tasks.	Attack description, bug reports, social media content, hardware documentation, etc.	[80] [3] [316] [103] [197]
Initial data segmentation	Categorize data into distinct groups as needed.	Split data into sentences or words.	[148] [198] [4]
Unqualified data deletion	Delete invalid text data according to the specified rules.	Remove certain symbols and words (rare words, stop words, etc.), or convert all content to lowercase.	[78] [33] [5]
Text representation	Token-based text representation.	Tokenize the texts, sentences, or words into tokens.	[4] [197]
Data segmentation	Divide the dataset into training, validation, and testing subsets for model training, parameter tuning, and performance evaluation.	Partition the dataset based on specific criteria, which may include division into training, validation, or testing subsets.	[244] [296] [178]

Data preprocessing techniques for code-based datasets. We outline the preprocessing techniques utilized for code-based datasets, often comprising five key steps. The initial step involves **data extraction**, retrieving relevant code snippets from diverse sources. Depending on the research task's needs [202, 270], snippets may be extracted at different levels of detail, ranging from individual lines, methods, or functions to entire code files or projects. To prevent bias and redundancy during training, the next step removes **duplicated instances** by identifying and eliminating them from the dataset [342, 349], enhancing diversity and uniqueness. **Unqualified data deletion** follows, removing snippets that don't meet predefined quality standards (e.g., based on size, complexity, or presence of obvious hints) to ensure relevance to the security task and avoid noise [84, 338]. **Code representation** converts snippets into suitable formats for LLM processing, typically involving tokenization for security tasks [318]. Finally, **data splitting** divides the preprocessed dataset into training, validation, and testing subsets [328]. Training sets are used to train the LLM, validation sets help tune hyperparameters, and testing sets assess model performance on unseen data.

Data preprocessing techniques for text-based datasets. As depicted in Table 10, preprocessing text-based datasets involves five steps, with minor differences compared to code-based datasets. The process begins with data extraction, carefully retrieving text from various sources such as bug reports [78], program documentation [316], hardware documentation [197], and social media content [103]. This initial phase ensures the dataset encompasses a range of task-specific textual information. After data extraction, the text undergoes segmentation tailored to the specific research task's needs. Segmentation may involve breaking text into sentences or further dividing it into individual words for analysis [4, 197]. Subsequent preprocessing operations standardize and clean the text, typically involving the removal of specific symbols, stop words, and special characters [4, 198]. This standardized textual format facilitates effective processing by LLMs. To address bias and redundancy in the dataset, this step enhances dataset diversity, aiding the model's generalization to new inputs [148]. Data tokenization is essential for constructing LLM inputs, where text is tokenized into smaller units like words or subwords to facilitate feature learning [4]. Finally, the preprocessed dataset is divided into subsets, typically comprising training, validation, and testing sets.

6.4 Data Augmentation using LLMs

A recurring challenge in applying LLMs to cybersecurity tasks is the scarcity of high-quality, labeled data. Datasets for vulnerabilities, malware, specific attack types, or other security events can be difficult to obtain in sufficient volume and diversity, hindering the training of robust models. To address this bottleneck, a number of studies have explored leveraging the generative capabilities of LLMs themselves for data augmentation [67]. The overarching goal is to enhance dataset size, diversity, and balance, thereby improving the robustness and generalization performance of security models trained (typically via fine-tuning) on the augmented data.

These LLM-based data augmentation strategies manifest in various forms depending on the task and data type. One common approach involves generating synthetic code samples used to enlarge fine-tuning datasets. This includes creating new examples of vulnerable code, such as RealVul using code transformations based on PHP vulnerability patterns [35], prompting LLMs with CWE details to generate pairs of vulnerable and non-vulnerable functions [290], implanting specific bug types into correct code snippets to create debugging benchmarks [362], or generating formally verified vulnerable and benign code snippets [281]. LLMs are also used to create specific malicious entities, for instance, generating diverse synthetic variations of malicious NPM packages based on known seeds to improve the fine-tuning of malware detection training [337]. In the textual domain, LLMs synthesize relevant data often used for fine-tuning or as few-shot examples, such as examples of harmful online discourse [194], natural language repair suggestions ("oracles") generated by prompting an LLM with vulnerable code and patches which then form a corpus for fine-tuning suggestion models [346], concise vulnerability explanations derived from patch features used for training auxiliary tasks during multi-task fine-tuning [69], or richer textual features like the explanatory 'debate text' produced by EXPLAINHM to fine-tune a classifier for harmful meme classification [168]. In a related application, LLMs assist in augmenting or constructing the knowledge bases essential for security tools, exemplified by VULTURE, which uses LLMs to analyze CVEs and commits for building a comprehensive third-party library vulnerability database [325] used by the downstream detection tool.

RQ4 - Summary

(1) Our analysis of data sources reveals a significant reliance on open-source, collected, and constructed datasets, reflecting a persistent challenge of data scarcity. A critical insight is the simultaneous disconnect from industrial practice, evidenced by the extreme scarcity of industrial datasets (used in only 1% of papers), which raises questions about the real-world applicability of current models.
(2) We categorize datasets into three primary types: code-based, text-based, and combined. While code-based data is predominant, a key trend is the growing use of combined datasets, which pair code with natural language descriptions to provide crucial context and enhance the semantic understanding of LLMs.
(3) To address the data scarcity challenge, a notable insight is the trend of leveraging LLMs for data augmentation. Our review shows researchers are increasingly using LLMs to generate synthetic code samples, malicious entities, and diverse textual data to enhance the size and quality of training sets.
(4) We summarize the common data preprocessing pipeline applied across studies, which includes essential steps such as data extraction, cleaning and deduplication, converting data into suitable representations for LLMs, and segmentation into training, validation, and testing sets.

7 THREATS TO VALIDITY

Paper retrieval omissions. One significant potential risk is the possibility of overlooking relevant papers during the search process. While collecting papers on LLM4Security tasks from various publishers, there is a risk of missing out on papers with incomplete abstracts, lacking cybersecurity tasks or LLM keywords. To address this issue, we employed a comprehensive approach that combines manual searching, automated searching, and snowballing techniques to minimize the chances of overlooking relevant papers as much as possible. We extensively searched for LLM papers related to security tasks in three top security conferences, extracting authoritative and comprehensive security task and LLM keywords for manual searching. Additionally, we conducted automated searches using carefully crafted keyword search strings on seven widely used publishing platforms. Furthermore, to further expand our search results, we employed both forward and backward snowballing techniques.

Bias of research selection. The selection of studies carries inherent limitations and potential biases. Initially, we established criteria for selecting papers through a combination of automated and manual steps, followed by manual validation based on Quality Assessment Criteria (QAC). However, incomplete or ambiguous information in BibTeX records may result in mislabeling of papers during the automated selection process. To address this issue, papers that cannot be conclusively excluded require manual validation. However, the manual validation stage may be subject to biases in researchers' subjective judgments, thereby affecting the accuracy of assessing paper quality. To mitigate these issues, we enlisted two experienced reviewers from the fields of cybersecurity and LLM to conduct a secondary review of the research selection results. This step aims to enhance the accuracy of paper selection and reduce the chances of omission or misclassification. By implementing these measures, we strive to ensure the accuracy and integrity of the selected papers, minimize the impact of selection biases, and enhance the reliability of the systematic literature review. Additionally, we provide a replication package for further examination by others.

8 CHALLENGES AND OPPORTUNITIES

8.1 Challenges

8.1.1 Challenges in LLM Applicability.

Model size and deployment. The size of LLMs have seen significant growth over time, escalating from 117M parameters for GPT-1 to 1.5B parameters for GPT-2, and further to 175B parameters for GPT-3 [331]. Models with billions or even trillions of parameters present substantial challenges in terms of storage, memory, and computational demands [81]. This can potentially impede the deployment of LLMs, particularly in scenarios where developers lack access to potent GPUs or TPUs, especially in resource-constrained environments necessitating real-time deployment. CodeBERT [83] emerged in 2019 as a pre-trained model featuring 125M parameters and a model size of 476MB. Recent models like Codex [44] and CodeGen [213] have surpassed 100 billion parameters, with model sizes exceeding 100GB. Larger sizes entail more computational resources and higher time costs. For instance, training the GPT-Neox-20B model [24] mandates 825GB of raw text data and deployment on 8 NVIDIA A100-SXM4-40GB graphics processing units (GPUs). Each GPU comes with a price tag of over $6,000, and the training duration spans 1,830 hours or roughly 76 days. These instances underscore the substantial computational costs linked with training LLMs. Additionally, these platforms entail notable energy expenses, with LLM-based platforms projected to markedly amplify energy consumption [245]. Some vendors like OpenAI and Google provide online APIs for LLMs to alleviate user usage costs, while researchers explore methods to curtail LLM scale. Hsieh et al. [115] proposed step-by-step distillation to diminish the data and

model size necessary for LLM training, with their findings showcasing that a T5 model with only 770MB surpassed a 540B PaLM.

Data Quality, Availability, and Security Challenges. In Section 6, we conducted an extensive examination of the datasets and data preprocessing procedures employed in the 118 studies. Our analysis unveiled the heavy reliance of LLMs on a diverse array of datasets for training and fine-tuning. The findings underscore the challenge of data scarcity encountered by LLMs when tackling security tasks. The quality, diversity, and volume of data directly influence the performance and generalization capabilities of these models. Given their scale, LLMs typically necessitate substantial data volumes to capture nuanced distinctions, yet acquiring such data poses significant challenges. Many specific security tasks suffer from a dearth of high-quality and robust publicly available datasets. Relying on limited or biased datasets may result in models inheriting these biases, leading to skewed or inaccurate predictions. Furthermore, there is a concern regarding the risk of benchmark data contamination, where existing research may involve redundant filtering of native data, potentially resulting in overlap between training and testing datasets, thus inflating performance metrics [154]. Additionally, we raise serious apprehensions regarding the inclusion of personally private information, such as phone numbers and email addresses, in training corpora when LLMs are employed for information and content security tasks, which precipitate privacy breaches during the prompting process [73].

8.1.2 Challenges in LLM Generalization Ability. The generalization capability of LLMs pertains to their ability to consistently and accurately execute tasks across diverse tasks, datasets, or domains beyond their training environment. Despite undergoing extensive pre-training on large datasets to acquire broad knowledge, the absence of specialized expertise can present challenges when LLMs encounter tasks beyond their pre-training scope, especially in the cybersecurity domain. As discussed in Section 4, we explored the utilization of LLMs in 21 security tasks spanning five security domains. We observed substantial variations in the context and semantics of code or documents across different domains and task specifications. To ensure LLMs demonstrate robust generalization, meticulous fine-tuning, validation, and continuous feedback loops on datasets from various security tasks are imperative. Without these measures, there's a risk of models overfitting to their training data, thus limiting their efficacy in diverse real-world scenarios.

8.1.3 Challenges in LLM Interpretability, Trustworthiness, and Ethical Usage. Ensuring interpretability and trustworthiness is paramount when integrating LLMs into security tasks, particularly given the sensitive nature of security requirements and the need for rigorous scrutiny of model outputs. The challenge lies in comprehending how these models make decisions, as the black-box nature of LLMs often impedes explanations for why or how specific outputs or recommendations are generated for security needs. While LLM self-explanations are often post-hoc rationalizations rather than reflections of internal workings [87, 253], research explores methods to enhance their transparency and utility in security contexts. Techniques include promoting step-by-step reasoning during generation (e.g., Chain-of-Thought) which can improve output quality [87, 146], grounding explanations in source data using RAG or citation mechanisms, visualizing attention weights as proxies [72], defining structured evaluation criteria to constrain the LLM's reasoning process, and applying established XAI methods like LIME and SHAP to explain the behavior of models used within security workflows [167]. Furthermore, assessing the reliability of LLM judgments can be aided by estimating output uncertainty or checking for consistency across multiple generated responses [287]. A promising approach involves using LLMs specifically as 'explanation translators,' taking technical outputs from XAI tools (like feature importance scores from LIME/SHAP) and converting them into accessible, natural language descriptions tailored for end-users, as demonstrated in phishing detection [167].

Recent research [232, 298] has also underscored that artificial intelligence-generated content (AIGC) introduces additional security risks, including privacy breaches, dissemination of forged information, and the generation of vulnerable code. The absence of interpretability and trustworthiness can breed user uncertainty and reluctance, as stakeholders may hesitate to rely on LLMs for security tasks without a clear understanding of their decision-making process or adherence to security requirements. Establishing trust in LLMs necessitates further development and application of technologies and tools that offer deeper insights into model internals and output rationale, empowering developers and users to comprehend and verify the reasoning [87]. Improving interpretability and trustworthiness through such methods can ultimately foster the widespread adoption of cost-effective automation in the cybersecurity domain, fostering more efficient and effective security practices. However, many LLMs lack open-source availability, and questions persist regarding the data on which they were trained, as well as the quality, sources, and ownership of the training data, raising concerns about ownership regarding LLM-generated tasks. Moreover, there is the looming threat of various adversarial attacks, including tactics to guide LLMs to circumvent security measures and expose their original training data [60].

8.1.4 Limitations in Modern Cryptanalysis. A significant challenge to the broad applicability of LLMs is evident in the field of modern cryptanalysis. Current LLMs operate as probabilistic models, proficient at identifying statistical patterns within extensive textual datasets [98]. This operational paradigm inherently conflicts with the deterministic, precise, and mathematically rigorous demands of cryptanalysis, which typically necessitates profound insights into areas like number theory and algebra to devise novel algorithms for computationally hard problems [26, 273]. LLMs demonstrate notable limitations in executing complex multi-step mathematical computations, engaging in formal logical deduction, and managing abstract conceptualizations, often leading to errors or diminished performance when faced with irrelevant contextual information [11, 106]. Moreover, the data-centric learning approach of LLMs is fundamentally misaligned with the objective of discovering new attack methodologies for contemporary ciphers, primarily because datasets illustrating successful exploits against such robust systems are, by design, unavailable for training. The tendency of LLMs to generate specious yet plausible information (i.e., hallucinations) further compromises their utility in cryptanalytic applications where exactitude is non-negotiable [352]. Thus, the core architecture of LLMs as sophisticated language processors, rather than as deterministic logical inference engines, establishes a fundamental incompatibility with the stringent requirements for breaking modern cryptographic systems.

8.2 Opportunities

8.2.1 Improvement of LLM4Security.

Training models for security tasks. Deciding between commercially available pre-trained models like GPT-4 [219] and open-source frameworks such as T5 [241] or LLaMa [283] presents a nuanced array of choices for tailoring tasks to individual or organizational needs. The distinction between these approaches lies in the level of control and customization they offer. Pre-trained models like GPT-4 are generally not intended for extensive retraining but allow for quick adaptation to specific tasks with limited data, thus reducing computational overhead. Conversely, frameworks like T5 offer an open-source platform for broader customization. While they undergo pre-training, researchers often modify the source code and retrain these models on their own large-scale datasets to meet specific task requirements [110]. This process demands substantial computational resources, resulting in higher resource allocation and costs, but provides the advantage of creating highly specialized models tailored to specific domains. Therefore, the main trade-off lies between

the user-friendly nature and rapid deployment offered by models like GPT-4 and the extensive task customization capabilities and increased computational demands associated with open-source frameworks like T5.

Inter-model interaction of LLMs. Our examination indicates that LLMs have progressed significantly in tackling various security challenges. However, as security tasks become more complex, there's a need for more sophisticated and tailored solutions. As outlined in Section 5, one promising avenue is collaborative model interaction through external augmentation methods. This approach involves integrating multiple LLMs [330] or combining LLMs with specialized machine learning models [33, 276] to improve task efficiency while simplifying complex steps. By harnessing the strengths of different models collectively, we anticipate that LLMs can deliver more precise and higher-quality outcomes for intricate security tasks.

Impact and applications of ChatGPT. In recent academic research, ChatGPT has garnered considerable attention, appearing in over half of the 185 papers we analyzed. It has been utilized to tackle specific security tasks, highlighting its growing influence and acceptance in academia. Researchers have favored ChatGPT due to its computational efficiency, versatility across tasks, and potential cost-effectiveness compared to other LLMs and LLM-based applications [150]. Beyond generating task solutions, ChatGPT promotes collaboration, signaling a broader effort to integrate advanced natural language understanding into traditional cybersecurity practices [61, 234]. By closely examining these trends, we can anticipate pathways for LLMs and applications like ChatGPT to contribute to more robust, efficient, and collaborative cybersecurity solutions. These insights highlight the transformative potential of LLMs in shaping the future cybersecurity landscape.

8.2.2 Enhancing LLM's Performance in Existing Security Tasks.

External retrieval and tools for LLM. LLMs have demonstrated impressive performance across diverse security tasks, but they are not immune to inherent limitations, including a lack of domain expertise [138], a tendency to generate hallucinations [352], weak mathematical capabilities, and a lack of interpretability. Therefore, a feasible approach to enhancing their capabilities is to enable them to interact with the external world, acquiring knowledge in various forms and manners to improve the factualness and rationality of generated security task solutions. One viable solution is to provide external knowledge bases for LLMs, augmenting content generation with retrieval-based methods to retrieve task-relevant data for LLM outputs [70, 92]. Another approach is to incorporate external specialized tools to provide real-time interactive feedback to guide LLMs [14, 17], combining the results of specialized analytical tools to steer LLMs towards robust and consistent security task solutions. We believe that incorporating external retrieval and tools is a competitive choice for improving the performance of LLM4Security.

Addressing challenges in specific domains. Numerous cybersecurity domains, such as network security and hardware security, encounter a dearth of open-source datasets, impeding the integration of LLMs into these specialized fields [271]. Future endeavors may prioritize the development of domain-specific datasets and the refinement of LLMs to address the distinctive challenges and nuances within these domains. Collaborating with domain experts and practitioners is crucial for gathering relevant data, and fine-tuning LLMs with this data can improve their effectiveness and alignment with each domain's specific requirements. This collaborative approach helps LLMs address real-world challenges across different cybersecurity domains [31].

8.2.3 Expanding LLM's Capabilities in More Security Domains.

Integrating new input formats. In our research, we noticed that LLMs in security tasks typically use input formats from code-based and text-based datasets. The introduction of new input formats based on natural language, like voice and images, as well as multimodal inputs such as video demonstrations, presents an opportunity to enhance LLMs'

ability to understand and process various user needs [335]. Integrating speech can improve user-model interaction, allowing for more natural and context-rich communication. Images can visually represent security task processes and requirements, providing LLMs with additional perspectives. Moreover, multimodal inputs combining text, audio, and visuals can offer a more comprehensive contextual understanding, leading to more accurate and contextually relevant security solutions.

Expanding LLM applications. We noticed that LLMs have received significant attention in the domain of software and system security. This domain undoubtedly benefits from the text and code parsing capabilities of LLMs, leading to tasks such as vulnerability detection, program fuzzing, and others. Currently, the applications of LLMs in domains such as hardware security and blockchain security remain relatively limited, and specific security tasks in certain domains have not yet been explored by researchers using LLMs. This presents an important opportunity: by extending the use of LLMs to these underdeveloped domains, we can potentially drive the development of automated security solutions.

8.3 Roadmap

We present a roadmap for future progress in utilizing Large Language Models for Security (LLM4Security), while also acknowledging the reciprocal relationship and growing exploration of Security for Large Language Models (Security4LLM) from a high-level perspective.

Automating cybersecurity solutions. The quest for security automation encompasses the automated analysis of specific security scenario samples, multi-scenario security situational awareness, system security optimization, and the development of intelligent, tailored support for security operatives, which possesses context awareness and adaptability to individual needs. Leveraging the generative prowess of LLMs can aid security operatives in comprehending requirements better and crafting cost-effective security solutions, thus expediting security response times. Utilizing the natural language processing capabilities of LLMs to build security-aware tools enables more intuitive and responsive interactions with security operatives. Moreover, assisting security operatives in fine-tuning LLMs for specific security tasks can augment their precision and efficiency, tailoring automated workflows to cater to the distinct demands of diverse projects and personnel.

Incorporating security knowledge into LLMs. A key direction for the future is to integrate specialized security task solutions and knowledge from the cybersecurity domain into LLMs to overcome potential hallucinations and errors [3, 171]. This integration aims to enhance LLMs' ability to address security tasks, especially those requiring a significant amount of domain expertise, such as penetration testing [61, 278], hardware vulnerability detection [64], log analysis [141, 174], and more. Embedding rules and best practices from specific security domains into these models will better represent task requirements, enabling LLMs to generate robust and consistent security task solutions [52].

Security agent: integrating external augmentation and LLMs. We have witnessed the unprecedented potential of applying LLMs to solve security tasks, almost overturning traditional security task solutions in LLM4Security [47, 166, 316]. However, the inherent lack of domain-specific knowledge and hallucinations in LLMs restrict their ability to perceive task requirements or environments with high quality [352]. AI Agents are artificial entities that perceive the environment, make decisions, and take actions. Currently, they are considered the most promising tool for achieving the pursuit of achieving or surpassing human-level intelligence in specific domains [315]. We summarized the external enhancement techniques introduced in LLM4Security in Section 5, optimizing LLMs' performance in security tasks across multiple dimensions, including input, model, and output [47, 133, 231]. Security operators can specify specific external enhancement strategies for security tasks and integrate them with LLMs to achieve automated security AI agents with continuous interaction within the system.

Multimodal LLMs for security. In LLM4Security, all research inputs are based on textual language (text or code). With the rise of multimodal generative LLMs represented by models like Sora [220], we believe that future research in LLM4Security can expand to include multimodal inputs and outputs such as video, audio, and images to enhance LLMs' understanding and processing of security tasks. For example, when using LLMs as penetration testing tools, relevant images such as topology diagrams of the current network environment and screenshots of the current steps can be introduced as inputs. In addition, audio inputs (such as recordings of specific security incidents or discussions) can provide further background information for understanding security task requirements.

Security for Large Language Models (Security4LLM). LLMs have gained considerable traction in the security sector, showcasing their potential in security-related endeavors. Nonetheless, delving into the internal security assessment of LLMs themselves remains a pressing area for investigation [56, 122, 332]. The intricate nature of LLMs renders them vulnerable to a wide range of attacks, necessitating innovative strategies to fortify the models [60, 94, 182, 332, 363]. These vulnerabilities can be broadly categorized [332]. AI-inherent vulnerabilities stem from the machine learning aspects and include adversarial attacks like data poisoning (manipulating training data to compromise the model [289]) and backdoor attacks (embedding hidden triggers), as well as privacy-compromising inference attacks (attribute inference [267], membership inference [200]) and extraction attacks (recovering sensitive training data [36]). Instruction tuning attacks, such as jailbreaking (bypassing safety restrictions [60]) and prompt injection (manipulating prompts to elicit unintended behavior [97]), exploit the way LLMs process instructions. Non-AI inherent vulnerabilities may arise from the surrounding infrastructure, such as remote code execution (RCE) in integrated applications [179] or supply chain vulnerabilities involving third-party plugins [125].

Furthermore, LLMs can themselves be misused as tools for offensive purposes [332]). Their advanced generation capabilities can be exploited to create sophisticated phishing emails [109], generate malware [28], spread misinformation [40], or facilitate social engineering attacks [75]. User-level attacks are particularly prevalent due to the human-like reasoning abilities of LLMs [332]. Considering that the inputs and outputs for security tasks often involve security-sensitive data (such as system logs or vulnerability code in programs) [227, 235], the potential for misuse or data leakage poses significant cybersecurity risks.

Mitigating these threats requires a multi-faceted approach [332]. Defense strategies can be implemented at different stages: improving model architecture (e.g., considering capacity or sparsity [165]), during training (e.g., corpora cleaning [268], adversarial training [291], safe instruction tuning [22], applying differential privacy [165]), and during inference (e.g., instruction pre-processing [126], in-process malicious use detection [301], and output post-processing like self-critique [135]). An intriguing avenue for future research is to empower LLMs to autonomously detect and identify their own vulnerabilities. Specifically, efforts could focus on enabling LLMs to generate patches for their underlying code, thus bolstering their inherent security, rather than solely implementing program restrictions at the user interaction layer. Given this scenario, future research should adopt a balanced approach, striving to utilize LLMs for automating cost-effective completion of security tasks while simultaneously developing and evaluating robust techniques to safeguard the LLMs themselves. This dual focus is pivotal for fully harnessing the potential of LLMs in enhancing cybersecurity and ensuring compliance with cyber systems.

9 CONCLUSION

The rapid integration of LLMs is creating a paradigm shift in cybersecurity, particularly by revolutionizing how security is integrated into the software engineering lifecycle. In this systematic literature review of 185 seminal papers, we

have not only mapped the landscape of LLM4Security but also synthesized several key insights that highlight this transformation:

- **A Paradigm Shift from Analysis to Generation and Repair:** A primary contribution of LLMs to software engineering is moving beyond traditional defect detection to automated generation and repair. Decoder-only models, in particular, are being leveraged to create and validate patches for software vulnerabilities and bugs, representing a significant leap towards self-healing software.
- **Solving Data Scarcity through Generative Augmentation:** LLMs offer a novel solution to the long-standing problem of data scarcity in security research. Our review found a prominent trend of using LLMs to generate high-quality synthetic data—from vulnerable code snippets to textual repair suggestions—thereby creating richer and more diverse datasets for training robust security tools.
- **The Rise of Hybrid Intelligence Workflows:** The most effective applications of LLMs do not operate in isolation. Instead, they act as intelligent "brains" orchestrating traditional software engineering tools like static analyzers, fuzzers, and verifiers. This hybrid, neuro-symbolic approach, which often involves external augmentation and tool integration, is becoming the de facto standard for building reliable LLM-powered security solutions.
- **The Emergence of Autonomous Agents:** Beyond single-task automation, a forward-looking trend is the development of autonomous LLM-based agents capable of handling complex, multi-step security workflows. These agents, which can plan, use tools, and learn from interactions, are being applied to entire vulnerability lifecycles and penetration testing, pointing towards a future of autonomous security operations.

These insights are the culmination of our in-depth analysis guided by four Research Questions (RQs), which systematically examined the security tasks (RQ1), LLM models (RQ2), adaptation techniques (RQ3), and data handling practices (RQ4) in the field. By also outlining the critical challenges and providing a roadmap for future work, this survey serves as a valuable resource for researchers and practitioners aiming to navigate and contribute to the rapidly evolving intersection of large language models and cybersecurity.

REFERENCES

[1] 2023. Models. https://huggingface.co/models.

[2] Abdelrahman Abdallah and Adam Jatowt. 2024. Generator-Retriever-Generator Approach for Open-Domain Question Answering. arXiv:2307.11278 [cs.CL]

[3] Ehsan Aghaei and Ehab Al-Shaer. 2023. CVE-driven Attack Technique Prediction with Semantic Information Extraction and a Domain-specific Language Model. arXiv:2309.02785 [cs.CR]

[4] Ehsan Aghaei, Ehab Al-Shaer, Waseem Shadid, and Xi Niu. 2023. Automated CVE Analysis for Threat Prioritization and Impact Prediction. arXiv:2309.03040 [cs.CR]

[5] Ehsan Aghaei, Xi Niu, Waseem Shadid, and Ehab Al-Shaer. 2023. SecureBERT: A Domain-Specific Language Model for Cybersecurity. In *Security and Privacy in Communication Networks*, Fengjun Li, Kaitai Liang, Zhiqiang Lin, and Sokratis K. Katsikas (Eds.). Springer Nature Switzerland, Cham, 39–56.

[6] Rio Aguina-Kang, Maxim Gumin, Do Heon Han, Stewart Morris, Seung Jean Yoo, Aditya Ganeshan, R. Kenny Jones, Qiuhong Anna Wei, Kailiang Fu, and Daniel Ritchie. 2024. Open-Universe Indoor Scene Generation using LLM Program Synthesis and Uncurated Object Databases. arXiv:2403.09675 [cs.CV]

[7] Baleegh Ahmad, Benjamin Tan, Ramesh Karri, and Hammond Pearce. 2023. FLAG: Finding Line Anomalies (in code) with Generative AI. arXiv:2306.12643 [cs.CR]

[8] Baleegh Ahmad, Shailja Thakur, Benjamin Tan, Ramesh Karri, and Hammond Pearce. 2024. On Hardware Security Bug Code Fixes by Prompting Large Language Models. *IEEE Transactions on Information Forensics and Security* 19 (2024), 4043–4057. https://doi.org/10.1109/TIFS.2024.3374558

[9] Baleegh Ahmad, Shailja Thakur, Benjamin Tan, Ramesh Karri, and Hammond Pearce. 2024. On Hardware Security Bug Code Fixes by Prompting Large Language Models. *IEEE Transactions on Information Forensics and Security* 19 (2024), 4043–4057. https://doi.org/10.1109/TIFS.2024.3374558

[10] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified Pre-training for Program Understanding and Generation. arXiv:2103.06333 [cs.CL]

[11] Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. 2024. Large language models for mathematical reasoning: Progresses and challenges. *arXiv preprint arXiv:2402.00157* (2024).

[12] Tarek Ali and Panos Kostakos. 2023. HuntGPT: Integrating Machine Learning-Based Anomaly Detection and Explainable AI with Large Language Models (LLMs). arXiv:2309.16021 [cs.CR]

[13] Natasha Alkhatib, Maria Mushtaq, Hadi Ghauch, and Jean-Luc Danger. 2022. CAN-BERT do it? Controller Area Network Intrusion Detection System based on BERT Language Model. In *2022 IEEE/ACS 19th International Conference on Computer Systems and Applications (AICCSA)*. 1–8. https://doi.org/10.1109/AICCSA56895.2022.10017800

[14] Kamel Alrashedy and Abdullah Aljasser. 2024. Can LLMs Patch Security Issues? arXiv:2312.00024 [cs.CR]

[15] Ross J Anderson and Fabien AP Petitcolas. 1998. On the limits of steganography. *IEEE Journal on selected areas in communications* 16, 4 (1998), 474–481.

[16] M Anon. 2022. National vulnerability database. https://www.nist.gov/programs-projects/national-vulnerabilitydatabase-nvd.

[17] Jordi Armengol-Estapé, Jackson Woodruff, Chris Cummins, and Michael F.P. O'Boyle. 2024. SLaDe: A Portable Small Language Model Decompiler for Optimized Assembly. In *2024 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. 67–80. https://doi.org/10.1109/CGO57630.2024.10444788

[18] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. 2022. Constitutional AI: Harmlessness from AI Feedback. arXiv:2212.08073 [cs.CL]

[19] Atieh Bakhshandeh, Abdalsamad Keramatfar, Amir Norouzi, and Mohammad Mahdi Chekidehkhoun. 2023. Using ChatGPT as a Static Application Security Testing Tool. arXiv:2308.14434 [cs.CR]

[20] Luke A. Bauer, James K. Howes IV au2, Sam A. Markelon, Vincent Bindschaedler, and Thomas Shrimpton. 2022. Covert Message Passing over Public Internet Platforms Using Model-Based Format-Transforming Encryption. arXiv:2110.07009 [cs.CR]

[21] Nihar Bendre, Hugo Terashima Marín, and Peyman Najafirad. 2020. Learning from Few Samples: A Survey. arXiv:2007.15484 [cs.CV]

[22] Federico Bianchi, Mirac Suzgun, Giuseppe Attanasio, Paul Röttger, Dan Jurafsky, Tatsunori Hashimoto, and James Zou. 2024. Safety-Tuned LLaMAs: Lessons From Improving the Safety of Large Language Models that Follow Instructions. arXiv:2309.07875 [cs.CL] https://arxiv.org/abs/2309.07875

[23] Tristan Bilot, Nour El Madhoun, Khaldoun Al Agha, and Anis Zouaoui. 2023. A Survey on Malware Detection with Graph Representation Learning. arXiv:2303.16004 [cs.CR]

[24] Sid Black, Leo Gao, Phil Wang, Connor Leahy, and Stella Biderman. 2021. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow. https://api.semanticscholar.org/CorpusID:245758737

[25] Marcel Boehme, Cristian Cadar, and Abhik ROYCHOUDHURY. 2021. Fuzzing: Challenges and Reflections. *IEEE Software* 38, 3 (2021), 79–86. https://doi.org/10.1109/MS.2020.3016773

[26] Andrey Bogdanov, Takanori Isobe, and Elmar Tischhauser. 2016. Towards practical whitebox cryptography: optimizing efficiency and space hardness. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 126–158.

[27] Marcus Botacin. 2023. GPThreats-3: Is Automatic Malware Generation a Threat?. In *2023 IEEE Security and Privacy Workshops (SPW)*. 238–254. https://doi.org/10.1109/SPW59333.2023.00027

[28] Marcus Botacin. 2023. GPThreats-3: Is Automatic Malware Generation a Threat?. In *2023 IEEE Security and Privacy Workshops (SPW)*. 238–254. https://doi.org/10.1109/SPW59333.2023.00027

[29] Bernardo Breve, Gaetano Cimino, Giuseppe Desolda, Vincenzo Deufemia, and Annunziata Elefante. 2023. On the User Perception of Security Risks of TAP Rules: A User Study. In *End-User Development*, Lucio Davide Spano, Albrecht Schmidt, Carmen Santoro, and Simone Stumpf (Eds.). Springer Nature Switzerland, Cham, 162–179.

[30] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL]

[31] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. 2023. Sparks of Artificial General Intelligence: Early experiments with GPT-4. arXiv:2303.12712 [cs.CL]

[32] Yuchen Cai, Aashish Yadavally, Abhishek Mishra, Genesis Montejo, and Tien Nguyen. 2024. Programming Assistant for Exception Handling with CodeBERT. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 94, 13 pages. https://doi.org/10.1145/3597503.3639188

[33] Zijian Cai, Zhaoxuan Tan, Zhenyu Lei, Zifeng Zhu, Hongrui Wang, Qinghua Zheng, and Minnan Luo. 2024. LMBot: Distilling Graph Knowledge into Language Model for Graph-less Deployment in Twitter Bot Detection. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining* (Merida, Mexico) *(WSDM '24)*. Association for Computing Machinery, New York, NY, USA, 57–66. https://doi.org/10.1145/3616855.3635843

[34] Enrico Cambiaso and Luca Caviglione. 2023. Scamming the Scammers: Using ChatGPT to Reply Mails for Wasting Time and Resources. arXiv:2303.13521 [cs.CR]

[35] Di Cao, Yong Liao, and Xiuwei Shang. 2024. RealVul: Can We Detect Vulnerabilities in Web Applications with LLM?. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 8268–8282. https://doi.org/10.18653/v1/2024.emnlp-main.472

[36] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. 2021. Extracting training data from large language models. In *30th USENIX security symposium (USENIX Security 21)*. 2633–2650.

[37] Aaron Chan, Anant Kharkar, Roshanak Zilouchian Moghaddam, Yevhen Mohylevskyy, Alec Helyar, Eslam Kamal, Mohamed Elkamhawy, and Neel Sundaresan. 2023. Transformer-based Vulnerability Detection in Code at EditTime: Zero-shot, Few-shot, or Fine-tuning? arXiv:2306.01754 [cs.CR]

[38] Yiannis Charalambous, Norbert Tihanyi, Ridhi Jain, Youcheng Sun, Mohamed Amine Ferrag, and Lucas C. Cordeiro. 2023. A New Era in Software Security: Towards Self-Healing Software via Large Language Models and Formal Verification. arXiv:2305.14752 [cs.SE]

[39] P. V. Sai Charan, Hrushikesh Chunduri, P. Mohan Anand, and Sandeep K Shukla. 2023. From Text to MITRE Techniques: Exploring the Malicious Use of Large Language Models for Generating Cyber Attack Payloads. arXiv:2305.15336 [cs.CR]

[40] Canyu Chen and Kai Shu. 2024. Can LLM-Generated Misinformation Be Detected?. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=ccxD4mtkTU

[41] Chong Chen, Jianzhong Su, Jiachi Chen, Yanlin Wang, Tingting Bi, Jianxing Yu, Yanli Wang, Xingwei Lin, Ting Chen, and Zibin Zheng. 2024. When ChatGPT Meets Smart Contract Vulnerability Detection: How Far Are We? *ACM Trans. Softw. Eng. Methodol.* (Nov. 2024). https://doi.org/10.1145/3702973 Just Accepted.

[42] Guoqiang Chen, Huiqi Sun, Daguang Liu, Zhiqi Wang, Qiang Wang, Bin Yin, Lu Liu, and Lingyun Ying. 2025. ReCopilot: Reverse Engineering Copilot in Binary Analysis. arXiv:2505.16366 [cs.CR] https://arxiv.org/abs/2505.16366

[43] Jiachi Chen, Chong Chen, Jiang Hu, John Grundy, Yanlin Wang, Ting Chen, and Zibin Zheng. 2024. Identifying Smart Contract Security Issues in Code Snippets from Stack Overflow. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) *(ISSTA 2024)*. Association for Computing Machinery, New York, NY, USA, 1198–1210. https://doi.org/10.1145/3650212.3680353

[44] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG]

[45] Ping Chen, Lieven Desmet, and Christophe Huygens. 2014. A Study on Advanced Persistent Threats. In *Communications and Multimedia Security*, Bart De Decker and André Zúquete (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 63–72.

[46] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A Simple Framework for Contrastive Learning of Visual Representations. arXiv:2002.05709 [cs.LG]

[47] Tianyu Chen, Lin Li, Liuchuan Zhu, Zongyang Li, Guangtai Liang, Ding Li, Qianxiang Wang, and Tao Xie. 2023. VulLibGen: Identifying Vulnerable Third-Party Libraries via Generative Pre-Trained Model. arXiv:2308.04662 [cs.CR]

[48] Yufan Chen, Arjun Arunasalam, and Z. Berkay Celik. 2023. Can Large Language Models Provide Security & Privacy Advice? Measuring the Ability of LLMs to Refute Misconceptions. arXiv:2310.02431 [cs.HC]

[49] Yizheng Chen, Zhoujie Ding, Lamya Alowain, Xinyun Chen, and David 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* (Hong Kong, China) *(RAID '23)*. Association for Computing Machinery, New York, NY, USA, 654–668. https://doi.org/10.1145/3607199.3607242

[50] Yang Chen and Reyhaneh Jabbarvand. 2024. Neurosymbolic Repair of Test Flakiness. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) *(ISSTA 2024)*. Association for Computing Machinery, New York, NY, USA, 1402–1414. https://doi.org/10.1145/3650212.3680369

[51] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. 2023. Automatic Root Cause Analysis via Large Language Models for Cloud Incidents. arXiv:2305.15778 [cs.SE]

[52] Baijun Cheng, Kailong Wang, Ling Shi, Haoyu Wang, Yao Guo, Ding Li, and Xiangqun Chen. 2025. Enhancing Semantic Understanding in Pointer Analysis using Large Language Models. arXiv:2508.21454 [cs.SE] https://arxiv.org/abs/2508.21454

[53] Yiu Wai Chow, Luca Di Grazia, and Michael Pradel. 2024. PyTy: Repairing Static Type Errors in Python. *CoRR* abs/2401.06619 (2024). https://doi.org/10.48550/ARXIV.2401.06619 arXiv:2401.06619

[54] Yiu Wai Chow, Max Schäfer, and Michael Pradel. 2023. Beware of the Unexpected: Bimodal Taint Analysis. arXiv:2301.10545 [cs.SE]

[55] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. PaLM: Scaling Language Modeling with Pathways. arXiv:2204.02311 [cs.CL]

[56] Badhan Chandra Das, M. Hadi Amini, and Yanzhao Wu. 2025. Security and Privacy Challenges of Large Language Models: A Survey. *ACM Comput. Surv.* 57, 6, Article 152 (Feb. 2025), 39 pages. https://doi.org/10.1145/3712001

[57] Isaac David, Liyi Zhou, Kaihua Qin, Dawn Song, Lorenzo Cavallaro, and Arthur Gervais. 2023. Do you still need a manual smart contract audit? arXiv:2306.12338 [cs.CR]

[58] Gabriel de Jesus Coelho da Silva and Carlos Becker Westphall. 2024. A Survey of Large Language Models in Cybersecurity. arXiv:2402.16968 [cs.CR]

[59] 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, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948 [cs.CL] https://arxiv.org/abs/2501.12948

[60] Gelei Deng, Yi Liu, Yuekang Li, Kailong Wang, Ying Zhang, Zefeng Li, Haoyu Wang, Tianwei Zhang, and Yang Liu. 2024. MASTERKEY: Automated Jailbreaking of Large Language Model Chatbots. In *Proceedings 2024 Network and Distributed System Security Symposium (NDSS 2024)*. Internet Society. https://doi.org/10.14722/ndss.2024.24188

[61] Gelei Deng, Yi Liu, Victor Mayoral-Vilches, Peng Liu, Yuekang Li, Yuan Xu, Tianwei Zhang, Yang Liu, Martin Pinzger, and Stefan Rass. 2024. PentestGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, Philadelphia, PA, 847–864. https://www.usenix.org/conference/usenixsecurity24/presentation/deng

[62] Yinlin Deng, Chunqiu Steven Xia, Haoran Peng, Chenyuan Yang, and Lingming Zhang. 2023. Large Language Models Are Zero-Shot Fuzzers: Fuzzing Deep-Learning Libraries via Large Language Models. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis* (<conf-loc>, <city>Seattle</city>, <state>WA</state>, <country>USA</country>, </conf-loc>) *(ISSTA 2023)*. Association for Computing Machinery, New York, NY, USA, 423–435. https://doi.org/10.1145/3597926.3598067

[63] Yinlin Deng, Chunqiu Steven Xia, Chenyuan Yang, Shizhuo Dylan Zhang, Shujing Yang, and Lingming Zhang. 2024. Large Language Models are Edge-Case Generators: Crafting Unusual Programs for Fuzzing Deep Learning Libraries. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 70, 13 pages. https://doi.org/10.1145/3597503.3623343

[64] Ghada Dessouky, David Gens, Patrick Haney, Garrett Persyn, Arun Kanuparthi, Hareesh Khattri, Jason M. Fung, Ahmad-Reza Sadeghi, and Jeyavijayan Rajendran. 2019. Hardfails: insights into software-exploitable hardware bugs. In *Proceedings of the 28th USENIX Conference on Security Symposium* (Santa Clara, CA, USA) *(SEC'19)*. USENIX Association, USA, 213–230.

[65] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL]

[66] Dinil Mon Divakaran and Sai Teja Peddinti. 2024. LLMs for Cyber Security: New Opportunities. arXiv:2404.11338 [cs.CR]

[67] Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. 2020. Fine-Tuning Pretrained Language Models: Weight Initializations, Data Orders, and Early Stopping. arXiv:2002.06305 [cs.CL]

[68] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, Lei Li, and Zhifang Sui. 2023. A Survey on In-context Learning. arXiv:2301.00234 [cs.CL]

[69] Xiaohu Du, Ming Wen, Jiahao Zhu, Zifan Xie, Bin Ji, Huijun Liu, Xuanhua Shi, and Hai Jin. 2024. Generalization-Enhanced Code Vulnerability Detection via Multi-Task Instruction Fine-Tuning. In *Findings of the Association for Computational Linguistics: ACL 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 10507–10521. https://doi.org/10.18653/v1/2024.findings-acl.625

[70] Yali Du and Zhongxing Yu. 2023. Pre-training Code Representation with Semantic Flow Graph for Effective Bug Localization. arXiv:2308.12773 [cs.SE]

[71] Yanrui Du, Sendong Zhao, Jiawei Cao, Ming Ma, Danyang Zhao, Fenglei Fan, Ting Liu, and Bing Qin. 2024. Towards secure tuning: Mitigating security risks arising from benign instruction fine-tuning. *arXiv preprint arXiv:2410.04524* (2024).

[72] Nour El Houda Dehimi and Zakaria Tolba. 2024. Attention Mechanisms in Deep Learning : Towards Explainable Artificial Intelligence. In *2024 6th International Conference on Pattern Analysis and Intelligent Systems (PAIS)*. 1–7. https://doi.org/10.1109/PAIS62114.2024.10541203

[73] El-Mahdi El-Mhamdi, Sadegh Farhadkhani, Rachid Guerraoui, Nirupam Gupta, Lê-Nguyên Hoang, Rafael Pinot, Sébastien Rouault, and John Stephan. 2023. On the Impossible Safety of Large AI Models. arXiv:2209.15259 [cs.LG]

[74] Jueon Eom, Seyeon Jeong, and Taekyoung Kwon. 2024. Fuzzing JavaScript Interpreters with Coverage-Guided Reinforcement Learning for LLM-Based Mutation. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) *(ISSTA 2024)*. Association for Computing Machinery, New York, NY, USA, 1656–1668. https://doi.org/10.1145/3650212.3680389

[75] Polra Victor Falade. 2023. Decoding the Threat Landscape: ChatGPT, FraudGPT, and WormGPT in Social Engineering Attacks. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* (Oct. 2023), 185–198. https://doi.org/10.32628/cseit2390533

[76] Zhiyu Fan, Xiang Gao, Martin Mirchev, Abhik Roychoudhury, and Shin Hwei Tan. 2023. Automated Repair of Programs from Large Language Models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 1469–1481. https://doi.org/10.1109/ICSE48619.2023.00128

[77] Chongzhou Fang, Ning Miao, Shaurya Srivastav, Jialin Liu, Ruoyu Zhang, Ruijie Fang, Asmita, Ryan Tsang, Najmeh Nazari, Han Wang, and Houman Homayoun. 2024. Large language models for code analysis: do LLMs really do their job?. In *Proceedings of the 33rd USENIX Conference on Security Symposium* (Philadelphia, PA, USA) *(SEC '24)*. USENIX Association, USA, Article 47, 18 pages.

[78] Sen Fang, Tao Zhang, Youshuai Tan, He Jiang, Xin Xia, and Xiaobing Sun. 2023. RepresentThemAll: A Universal Learning Representation of Bug Reports. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 602–614. https://doi.org/10.1109/ICSE48619.2023.00060

[79] Weike Fang, Zhejian Zhou, Junzhou He, and Weihang Wang. 2024. StackSight: Unveiling webassembly through large language models and neurosymbolic chain-of-thought decompilation. In *Proceedings of the 41st International Conference on Machine Learning* (Vienna, Austria) *(ICML'24)*. JMLR.org, Article 521, 19 pages.

[80] Reza Fayyazi and Shanchieh Jay Yang. 2023. On the Uses of Large Language Models to Interpret Ambiguous Cyberattack Descriptions. arXiv:2306.14062 [cs.AI]

[81] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. arXiv:2101.03961 [cs.LG]

[82] Sidong Feng and Chunyang Chen. 2024. Prompting Is All You Need: Automated Android Bug Replay with Large Language Models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 67, 13 pages. https://doi.org/10.1145/3597503.3608137

[83] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. arXiv:2002.08155 [cs.CL]

[84] Mohamed Amine Ferrag, Ammar Battah, Norbert Tihanyi, Merouane Debbah, Thierry Lestable, and Lucas C. Cordeiro. 2023. SecureFalcon: The Next Cyber Reasoning System for Cyber Security. arXiv:2307.06616 [cs.CR]

[85] Mohamed Amine Ferrag, Mthandazo Ndhlovu, Norbert Tihanyi, Lucas C. Cordeiro, Merouane Debbah, Thierry Lestable, and Narinderjit Singh Thandi. 2024. Revolutionizing Cyber Threat Detection With Large Language Models: A Privacy-Preserving BERT-Based Lightweight Model for IoT/IIoT Devices. *IEEE Access* 12 (2024), 23733–23750. https://doi.org/10.1109/ACCESS.2024.3363469

[86] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. 2018. An Introduction to Deep Reinforcement Learning. *Foundations and Trends® in Machine Learning* 11, 3–4 (2018), 219–354. https://doi.org/10.1561/2200000071

[87] Vincent Freiberger, Arthur Fleig, and Erik Buchmann. 2025. Explainable AI in Usable Privacy and Security: Challenges and Opportunities. *arXiv preprint arXiv:2504.12931* (2025).

[88] Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen tau Yih, Luke Zettlemoyer, and Mike Lewis. 2023. InCoder: A Generative Model for Code Infilling and Synthesis. arXiv:2204.05999 [cs.SE]

[89] Jingzhou Fu, Jie Liang, Zhiyong Wu, and Yu Jiang. 2024. Sedar: Obtaining High-Quality Seeds for DBMS Fuzzing via Cross-DBMS SQL Transfer. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 146, 12 pages. https://doi.org/10.1145/3597503.3639210

[90] Michael Fu, Chakkrit Tantithamthavorn, Trung Le, Van Nguyen, and Dinh Phung. 2022. VulRepair: a T5-based automated software vulnerability repair. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (<conf-loc>, <city>Singapore</city>, <country>Singapore</country>, </conf-loc>) *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 935–947. https://doi.org/10.1145/3540250.3549098

[91] Yu Gai, Liyi Zhou, Kaihua Qin, Dawn Song, and Arthur Gervais. 2023. Blockchain Large Language Models. arXiv:2304.12749 [cs.CR]

[92] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997 [cs.CL]

[93] Arthur Gervais and Liyi Zhou. 2025. AI Agent Smart Contract Exploit Generation. *arXiv preprint arXiv:2507.05558* (2025).

[94] David Glukhov, Ilia Shumailov, Yarin Gal, Nicolas Papernot, and Vardan Papyan. 2023. LLM Censorship: A Machine Learning Challenge or a Computer Security Problem? arXiv:2307.10719 [cs.AI]

[95] Vasudev Gohil, Matthew DeLorenzo, Veera Vishwa Achuta Sai Venkat Nallam, Joey See, and Jeyavijayan Rajendran. 2025. LLMPirate: LLMs for Black-box Hardware IP Piracy. In *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/llmpirate-llms-for-black-box-hardware-ip-piracy/

[96] Google. 2023. Bard. https://Bard.google.com.

[97] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. arXiv:2302.12173 [cs.CR] https://arxiv.org/abs/2302.12173

[98] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, et al. 2024. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594* (2024).

[99] An Guo, Yuan Zhou, Haoxiang Tian, Chunrong Fang, Yunjian Sun, Weisong Sun, Xinyu Gao, Anh Tuan Luu, Yang Liu, and Zhenyu Chen. 2024. SoVAR: Build Generalizable Scenarios from Accident Reports for Autonomous Driving Testing. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento, CA, USA) *(ASE '24)*. Association for Computing Machinery, New York, NY, USA, 268–280. https://doi.org/10.1145/3691620.3695037

[100] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. GraphCodeBERT: Pre-training Code Representations with Data Flow. arXiv:2009.08366 [cs.SE]

[101] Xiao Han, Shuhan Yuan, and Mohamed Trabelsi. 2023. LogGPT: Log Anomaly Detection via GPT. arXiv:2309.14482 [cs.LG]

[102] Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. 2024. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608* (2024).

[103] Hans W. A. Hanley and Zakir Durumeric. 2023. Twits, Toxic Tweets, and Tribal Tendencies: Trends in Politically Polarized Posts on Twitter. arXiv:2307.10349 [cs.SI]

[104] Andreas Happe, Aaron Kaplan, and Jürgen Cito. 2023. Evaluating LLMs for Privilege-Escalation Scenarios. arXiv:2310.11409 [cs.CR]

[105] Thomas Hartvigsen, Saadia Gabriel, Hamid Palangi, Maarten Sap, Dipankar Ray, and Ece Kamar. 2022. ToxiGen: A Large-Scale Machine-Generated Dataset for Adversarial and Implicit Hate Speech Detection. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, Dublin, Ireland, 3309–3326. https://doi.org/10.18653/v1/2022.acl-long.234

[106] Alex Havrilla and Maia Iyer. 2024. Understanding the effect of noise in llm training data with algorithmic chains of thought. *arXiv preprint arXiv:2402.04004* (2024).

[107] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. DeBERTa: Decoding-enhanced BERT with Disentangled Attention. arXiv:2006.03654 [cs.CL]

[108] Fredrik Heiding, Bruce Schneier, Arun Vishwanath, Jeremy Bernstein, and Peter S. Park. 2024. Devising and Detecting Phishing Emails Using Large Language Models. *IEEE Access* 12 (2024), 42131–42146. https://doi.org/10.1109/ACCESS.2024.3375882

[109] Fredrik Heiding, Bruce Schneier, Arun Vishwanath, Jeremy Bernstein, and Peter S. Park. 2024. Devising and Detecting Phishing Emails Using Large Language Models. *IEEE Access* 12 (2024), 42131–42146. https://doi.org/10.1109/ACCESS.2024.3375882

[110] hiyouga. 2023. LLaMA Efficient Tuning. https://github.com/hiyouga/LLaMA-Efficient-Tuning.

[111] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. 2022. Training Compute-Optimal Large Language Models. arXiv:2203.15556 [cs.CL]

[112] Soneya Binta Hossain, Nan Jiang, Qiang Zhou, Xiaopeng Li, Wen-Hao Chiang, Yingjun Lyu, Hoan Nguyen, and Omer Tripp. 2024. A Deep Dive into Large Language Models for Automated Bug Localization and Repair. 1, FSE, Article 66 (July 2024), 23 pages. https://doi.org/10.1145/3660773

[113] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. arXiv:2308.10620 [cs.SE]

[114] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-Efficient Transfer Learning for NLP. arXiv:1902.00751 [cs.LG]

[115] Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller Model Sizes. arXiv:2305.02301 [cs.CL]

[116] Chuanbo Hu, Bin Liu, Xin Li, Yanfang Ye, and Minglei Yin. 2024. Knowledge-prompted ChatGPT: Enhancing drug trafficking detection on social media. *Inf. Manage.* 61, 6 (Sept. 2024), 12 pages. https://doi.org/10.1016/j.im.2024.104010

[117] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685 [cs.CL]

[118] Jie Hu, Qian Zhang, and Heng Yin. 2023. Augmenting Greybox Fuzzing with Generative AI. arXiv:2306.06782 [cs.CR]

[119] Peiwei Hu, Ruigang Liang, and Kai Chen. 2024. DeGPT: Optimizing Decompiler Output with LLM. *Proceedings 2024 Network and Distributed System Security Symposium* (2024). https://api.semanticscholar.org/CorpusID:267622140

[120] Sihao Hu, Tiansheng Huang, Fatih İlhan, Selim Furkan Tekin, and Ling Liu. 2023. Large Language Model-Powered Smart Contract Vulnerability Detection: New Perspectives. arXiv:2310.01152 [cs.CR]

[121] Kai Huang, Xiangxin Meng, Jian Zhang, Yang Liu, Wenjie Wang, Shuhao Li, and Yuqing Zhang. 2023. An empirical study on fine-tuning large language models of code for automated program repair. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1162–1174.

[122] Yue Huang, Lichao Sun, Haoran Wang, Siyuan Wu, Qihui Zhang, Yuan Li, Chujie Gao, Yixin Huang, Wenhan Lyu, Yixuan Zhang, Xiner Li, Hanchi Sun, Zhengliang Liu, Yixin Liu, Yijue Wang, Zhikun Zhang, Bertie Vidgen, Bhavya Kailkhura, Caiming Xiong, Chaowei Xiao, Chunyuan Li, Eric P. Xing, Furong Huang, Hao Liu, Heng Ji, Hongyi Wang, Huan Zhang, Huaxiu Yao, Manolis Kellis, Marinka Zitnik, Meng Jiang, Mohit Bansal, James Zou, Jian Pei, Jian Liu, Jianfeng Gao, Jiawei Han, Jieyu Zhao, Jiliang Tang, Jindong Wang, Joaquin Vanschoren, John Mitchell, Kai Shu, Kaidi Xu, Kai-Wei Chang, Lifang He, Lifu Huang, Michael Backes, Neil Zhenqiang Gong, Philip S. Yu, Pin-Yu Chen, Quanquan Gu, Ran Xu, Rex Ying, Shuiwang Ji, Suman Jana, Tianlong Chen, Tianming Liu, Tianyi Zhou, William Yang Wang, Xiang Li, Xiangliang Zhang, Xiao Wang, Xing Xie, Xun Chen, Xuyu Wang, Yan Liu, Yanfang Ye, Yinzhi Cao, Yong Chen, and Yue Zhao. 2024. Position: TrustLLM: Trustworthiness in Large Language Models. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*, Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (Eds.). PMLR, 20166–20270. https://proceedings.mlr.press/v235/huang24x.html

[123] Yuchao Huang, Junjie Wang, Zhe Liu, Yawen Wang, Song Wang, Chunyang Chen, Yuanzhe Hu, and Qing Wang. 2024. CrashTranslator: Automatically Reproducing Mobile Application Crashes Directly from Stack Trace. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 18, 13 pages. https://doi.org/10.1145/3597503.3623298

[124] Nasir Hussain, Haohan Chen, Chanh Tran, Philip Huang, Zhuohao Li, Pravir Chugh, William Chen, Ashish Kundu, and Yuan Tian. 2025. VulBinLLM: LLM-powered Vulnerability Detection for Stripped Binaries. arXiv:2505.22010 [cs.CR] https://arxiv.org/abs/2505.22010

[125] Umar Iqbal, Tadayoshi Kohno, and Franziska Roesner. 2025. *LLM Platform Security: Applying a Systematic Evaluation Framework to OpenAI's ChatGPT Plugins*. AAAI Press, 611–623.

[126] Neel Jain, Avi Schwarzschild, Yuxin Wen, Gowthami Somepalli, John Kirchenbauer, Ping yeh Chiang, Micah Goldblum, Aniruddha Saha, Jonas Geiping, and Tom Goldstein. 2023. Baseline Defenses for Adversarial Attacks Against Aligned Language Models. arXiv:2309.00614 [cs.LG] https://arxiv.org/abs/2309.00614

[127] Breier Jakub and Jana Branišová. 2017. A Dynamic Rule Creation Based Anomaly Detection Method for Identifying Security Breaches in Log Records. *Wireless Personal Communications* 94 (06 2017). https://doi.org/10.1007/s11277-015-3128-1

[128] Suhaima Jamal, Hayden Wimmer, and Iqbal H. Sarker. 2024. An improved transformer-based model for detecting phishing, spam and ham emails: A large language model approach. *Security and Privacy* 7, 5 (April 2024), 21 pages. https://doi.org/10.1002/spy2.402

[129] Jaeyoon Jeong, Insung Baek, Byungwoo Bang, Jun-Yeon Lee, Uiseok Song, and Seoung Bum Kim. 2025. FALL: Prior Failure Detection in Large Scale System Based on Language Model. *IEEE Trans. Dependable Secur. Comput.* 22, 1 (2025), 279–291. https://doi.org/10.1109/TDSC.2024.3396166

[130] Ziyou Jiang, Lin Shi, Guowei Yang, and Qing Wang. 2024. PatUntrack: Automated Generating Patch Examples for Issue Reports without Tracked Insecure Code. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento, CA, USA) *(ASE '24)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3691620.3694982

[131] Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How Can We Know What Language Models Know? arXiv:1911.12543 [cs.CL]

[132] Hongwei Jin, George Papadimitriou, Krishnan Raghavan, Pawel Zuk, Prasanna Balaprakash, Cong Wang, Anirban Mandal, and Ewa Deelman. 2024. Large Language Models for Anomaly Detection in Computational Workflows: From Supervised Fine-Tuning to In-Context Learning. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–17. https://doi.org/10.1109/SC41406.2024.00098

[133] Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. 2023. InferFix: End-to-End Program Repair with LLMs. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (San Francisco, CA, USA) *(ESEC/FSE 2023)*. Association for Computing Machinery, New York, NY, USA, 1646–1656. https://doi.org/10.1145/3611643.3613892

[134] Robert J. Joyce, Tirth Patel, Charles Nicholas, and Edward Raff. 2023. AVScan2Vec: Feature Learning on Antivirus Scan Data for Production-Scale Malware Corpora. arXiv:2306.06228 [cs.CR]

[135] Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. 2022. Language Models (Mostly) Know What They Know. arXiv:2207.05221 [cs.CL] https://arxiv.org/abs/2207.05221

[136] Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. 2023. Challenges and Applications of Large Language Models. arXiv:2307.10169 [cs.CL]

[137] Rahul Kande, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Shailja Thakur, Ramesh Karri, and Jeyavijayan Rajendran. 2024. (Security) Assertions by Large Language Models. *IEEE Transactions on Information Forensics and Security* 19 (2024), 4374–4389. https://doi.org/10.1109/TIFS.

2024.3372809

[138] Nikhil Kandpal, Haikang Deng, Adam Roberts, Eric Wallace, and Colin Raffel. 2023. Large Language Models Struggle to Learn Long-Tail Knowledge. arXiv:2211.08411 [cs.CL]

[139] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling Laws for Neural Language Models. arXiv:2001.08361 [cs.LG]

[140] Rabimba Karanjai, Sam Blackshear, Lei Xu, and Weidong Shi. 2025. A Multi-Agent Framework for Automated Vulnerability Detection and Repair in Solidity and Move Smart Contracts. arXiv:2502.18515 [cs.CR] https://arxiv.org/abs/2502.18515

[141] Egil Karlsen, Xiao Luo, Nur Zincir-Heywood, and Malcolm Heywood. 2023. Benchmarking Large Language Models for Log Analysis, Security, and Interpretation. arXiv:2311.14519 [cs.NI]

[142] Avishree Khare, Saikat Dutta, Ziyang Li, Alaia Solko-Breslin, Rajeev Alur, and Mayur Naik. 2023. Understanding the Effectiveness of Large Language Models in Detecting Security Vulnerabilities. arXiv:2311.16169 [cs.CR]

[143] Juhee Kim, Woohyuk Choi, and Byoungyoung Lee. 2025. Prompt Flow Integrity to Prevent Privilege Escalation in LLM Agents. arXiv:2503.15547 [cs.CR] https://arxiv.org/abs/2503.15547

[144] Barbara Kitchenham, O Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. 2009. Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology* 51, 1 (2009), 7–15.

[145] Takashi Koide, Hiroki Nakano, and Daiki Chiba. 2024. ChatPhishDetector: Detecting Phishing Sites Using Large Language Models. *IEEE Access* 12 (2024), 154381–154400. https://doi.org/10.1109/ACCESS.2024.3483905

[146] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Large Language Models are Zero-Shot Reasoners. arXiv:2205.11916 [cs.CL]

[147] He Kong, Die Hu, Jingguo Ge, Liangxiong Li, Tong Li, and Bingzhen Wu. 2025. Vulnbot: Autonomous penetration testing for a multi-agent collaborative framework. *arXiv preprint arXiv:2501.13411* (2025).

[148] Maxime Labonne and Sean Moran. 2023. Spam-T5: Benchmarking Large Language Models for Few-Shot Email Spam Detection. arXiv:2304.01238 [cs.CL]

[149] Md Tahmid Rahman Laskar, M Saiful Bari, Mizanur Rahman, Md Amran Hossen Bhuiyan, Shafiq Joty, and Jimmy Xiangji Huang. 2023. A Systematic Study and Comprehensive Evaluation of ChatGPT on Benchmark Datasets. arXiv:2305.18486 [cs.CL]

[150] Md Tahmid Rahman Laskar, M Saiful Bari, Mizanur Rahman, Md Amran Hossen Bhuiyan, Shafiq Joty, and Jimmy Xiangji Huang. 2023. A Systematic Study and Comprehensive Evaluation of ChatGPT on Benchmark Datasets. arXiv:2305.18486 [cs.CL]

[151] Tan Khang Le, Saba Alimadadi, and Steven Y. Ko. 2024. A Study of Vulnerability Repair in JavaScript Programs with Large Language Models. In *Companion Proceedings of the ACM Web Conference 2024* (Singapore, Singapore) *(WWW '24)*. Association for Computing Machinery, New York, NY, USA, 666–669. https://doi.org/10.1145/3589335.3651463

[152] Thanh Le-Cong, Duc-Minh Luong, Xuan Bach D. Le, David Lo, Nhat-Hoa Tran, Bui Quang-Huy, and Quyet-Thang Huynh. 2023. Invalidator: Automated Patch Correctness Assessment Via Semantic and Syntactic Reasoning. *IEEE Transactions on Software Engineering* 49, 6 (2023), 3411–3429. https://doi.org/10.1109/TSE.2023.3255177

[153] Jaehyung Lee, Kisun Han, and Hwanjo Yu. 2023. A Light Bug Triage Framework for Applying Large Pre-trained Language Model. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering* (<conf-loc>, <city>Rochester</city>, <state>MI</state>, <country>USA</country>, </conf-loc>) *(ASE '22)*. Association for Computing Machinery, New York, NY, USA, Article 3, 11 pages. https://doi.org/10.1145/3551349.3556898

[154] Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2022. Deduplicating Training Data Makes Language Models Better. arXiv:2107.06499 [cs.CL]

[155] Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The Power of Scale for Parameter-Efficient Prompt Tuning. arXiv:2104.08691 [cs.CL]

[156] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. arXiv:1910.13461 [cs.CL]

[157] Frank Li and Vern Paxson. 2017. A Large-Scale Empirical Study of Security Patches. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 2201–2215. https://doi.org/10.1145/3133956.3134072

[158] Haonan Li, Yu Hao, Yizhuo Zhai, and Zhiyun Qian. 2023. The Hitchhiker's Guide to Program Analysis: A Journey with Large Language Models. arXiv:2308.00245 [cs.SE]

[159] Kaixuan Li, Jian Zhang, Sen Chen, Han Liu, Yang Liu, and Yixiang Chen. 2024. PatchFinder: A Two-Phase Approach to Security Patch Tracing for Disclosed Vulnerabilities in Open-Source Software. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) *(ISSTA 2024)*. Association for Computing Machinery, New York, NY, USA, 590–602. https://doi.org/10.1145/3650212.3680305

[160] Lei Li, Yekun Chai, Shuohuan Wang, Yu Sun, Hao Tian, Ningyu Zhang, and Hua Wu. 2024. Tool-Augmented Reward Modeling. arXiv:2310.01045 [cs.CL]

[161] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour

Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. StarCoder: may the source be with you! arXiv:2305.06161 [cs.CL]

[162] Tsz-On Li, Wenxi Zong, Yibo Wang, Haoye Tian, Ying Wang, Shing-Chi Cheung, and Jeff Kramer. 2023. Nuances are the Key: Unlocking ChatGPT to Find Failure-Inducing Tests with Differential Prompting. arXiv:2304.11686 [cs.SE]

[163] Wenhao Li, Selvakumar Manickam, Yung wey Chong, and Shankar Karuppayah. 2025. PhishDebate: An LLM-Based Multi-Agent Framework for Phishing Website Detection. arXiv:2506.15656 [cs.CR] https://arxiv.org/abs/2506.15656

[164] Wenhao Li, Selvakumar Manickam, Yung wey Chong, and Shankar Karuppayah. 2025. PhishIntentionLLM: Uncovering Phishing Website Intentions through Multi-Agent Retrieval-Augmented Generation. arXiv:2507.15419 [cs.CR] https://arxiv.org/abs/2507.15419

[165] Xuechen Li, Florian Tramèr, Percy Liang, and Tatsunori Hashimoto. 2022. Large Language Models Can Be Strong Differentially Private Learners. arXiv:2110.05679 [cs.LG] https://arxiv.org/abs/2110.05679

[166] Hongliang Liang, Xiangyu Li, Da Xiao, Jie Liu, Yanjie Zhou, Aibo Wang, and Jin Li. 2024. Generative Pre-Trained Transformer-Based Reinforcement Learning for Testing Web Application Firewalls. *IEEE Transactions on Dependable and Secure Computing* 21, 1 (2024), 309–324. https://doi.org/10.1109/TDSC.2023.3252523

[167] Bryan Lim, Roman Huerta, Alejandro Sotelo, Anthonie Quintela, and Priyanka Kumar. 2025. EXPLICATE: Enhancing Phishing Detection through Explainable AI and LLM-Powered Interpretability. arXiv:2503.20796 [cs.CR] https://arxiv.org/abs/2503.20796

[168] Hongzhan Lin, Ziyang Luo, Wei Gao, Jing Ma, Bo Wang, and Ruichao Yang. 2024. Towards Explainable Harmful Meme Detection through Multimodal Debate between Large Language Models. In *Proceedings of the ACM Web Conference 2024* (Singapore, Singapore) *(WWW '24)*. Association for Computing Machinery, New York, NY, USA, 2359–2370. https://doi.org/10.1145/3589334.3645381

[169] Jie Lin and David Mohaisen. 2025. From Large to Mammoth: A Comparative Evaluation of Large Language Models in Vulnerability Detection. In *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/from-large-to-mammoth-a-comparative-evaluation-of-large-language-models-in-vulnerability-detection/

[170] Yu-Zheng Lin, Muntasir Mamun, Muhtasim Alam Chowdhury, Shuyu Cai, Mingyu Zhu, Banafsheh Saber Latibari, Kevin Immanuel Gubbi, Najmeh Nazari Bavarsad, Arjun Caputo, Avesta Sasan, Houman Homayoun, Setareh Rafatirad, Pratik Satam, and Soheil Salehi. 2023. HW-V2W-Map: Hardware Vulnerability to Weakness Mapping Framework for Root Cause Analysis with GPT-assisted Mitigation Suggestion. arXiv:2312.13530 [cs.CR]

[171] Chen Ling, Xujiang Zhao, Jiaying Lu, Chengyuan Deng, Can Zheng, Junxiang Wang, Tanmoy Chowdhury, Yun Li, Hejie Cui, Xuchao Zhang, Tianjiao Zhao, Amit Panalkar, Wei Cheng, Haoyu Wang, Yanchi Liu, Zhengzhang Chen, Haifeng Chen, Chris White, Quanquan Gu, Jian Pei, and Liang Zhao. 2023. Domain Specialization as the Key to Make Large Language Models Disruptive: A Comprehensive Survey. arXiv:2305.18703 [cs.CL]

[172] Bingchang Liu, Guozhu Meng, Wei Zou, Qi Gong, Feng Li, Min Lin, Dandan Sun, Wei Huo, and Chao Zhang. 2020. A large-scale empirical study on vulnerability distribution within projects and the lessons learned. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 1547–1559. https://doi.org/10.1145/3377811.3380923

[173] Fang Liu, Zhenwei Liu, Qianhui Zhao, Jing Jiang, Li Zhang, Zian Sun, Ge Li, Zhongqi Li, and Yuchi Ma. 2024. FastFixer: An Efficient and Effective Approach for Repairing Programming Assignments. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento, CA, USA) *(ASE '24)*. Association for Computing Machinery, New York, NY, USA, 669–680. https://doi.org/10.1145/3691620.3695062

[174] Jinyang Liu, Junjie Huang, Yintong Huo, Zhihan Jiang, Jiazhen Gu, Zhuangbin Chen, Cong Feng, Minzhi Yan, and Michael R. Lyu. 2023. Log-based Anomaly Detection based on EVT Theory with feedback. arXiv:2306.05032 [cs.SE]

[175] Muyang Liu, Ke Li, and Tao Chen. 2020. DeepSQLi: deep semantic learning for testing SQL injection. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Virtual Event, USA) *(ISSTA 2020)*. Association for Computing Machinery, New York, NY, USA, 286–297. https://doi.org/10.1145/3395363.3397375

[176] Puzhuo Liu, Chengnian Sun, Yaowen Zheng, Xuan Feng, Chuan Qin, Yuncheng Wang, Zhi Li, and Limin Sun. 2023. Harnessing the Power of LLM to Support Binary Taint Analysis. arXiv:2310.08275 [cs.CR]

[177] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. arXiv:2107.13586 [cs.CL]

[178] Ruitong Liu, Yanbin Wang, Haitao Xu, Zhan Qin, Yiwei Liu, and Zheng Cao. 2023. Malicious URL Detection via Pretrained Language Model Guided Multi-Level Feature Attention Network. arXiv:2311.12372 [cs.CR]

[179] Tong Liu, Zizhuang Deng, Guozhu Meng, Yuekang Li, and Kai Chen. 2024. Demystifying RCE Vulnerabilities in LLM-Integrated Apps. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (Salt Lake City, UT, USA) *(CCS '24)*. Association for Computing Machinery, New York, NY, USA, 1716–1730. https://doi.org/10.1145/3658644.3690338

[180] Xin Liu, Yuan Tan, Zhenghang Xiao, Jianwei Zhuge, and Rui Zhou. 2023. Not The End of Story: An Evaluation of ChatGPT-Driven Vulnerability Description Mappings. In *Findings of the Association for Computational Linguistics: ACL 2023*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 3724–3731. https://doi.org/10.18653/v1/2023.findings-acl.229

[181] Yang Liu, Jiahuan Cao, Chongyu Liu, Kai Ding, and Lianwen Jin. 2024. Datasets for Large Language Models: A Comprehensive Survey. arXiv:2402.18041 [cs.CL]

[182] Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, and Yang Liu. 2023. Prompt Injection attack against LLM-integrated Applications. *arXiv preprint arXiv:2306.05499* (2023).

[183] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692 [cs.CL]

[184] Ye Liu, Yue Xue, Daoyuan Wu, Yuqiang Sun, Yi Li, Miaolei Shi, and Yang Liu. 2025. PropertyGPT: LLM-driven Formal Verification of Smart Contracts through Retrieval-Augmented Property Generation. In *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/propertygpt-llm-driven-formal-verification-of-smart-contracts-through-retrieval-augmented-property-generation/

[185] You Lu, Yifan Tian, Yuyang Bi, Bihuan Chen, and Xin Peng. 2024. DiaVio: LLM-Empowered Diagnosis of Safety Violations in ADS Simulation Testing. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) *(ISSTA 2024)*. Association for Computing Machinery, New York, NY, USA, 376–388. https://doi.org/10.1145/3650212.3652135

[186] Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, et al. 2025. Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint arXiv:2503.21460* (2025).

[187] Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2023. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2308.08747* (2023).

[188] Kai Lv, Yuqing Yang, Tengxiao Liu, Qinghui Gao, Qipeng Guo, and Xipeng Qiu. 2023. Full Parameter Fine-tuning for Large Language Models with Limited Resources. arXiv:2306.09782 [cs.CL]

[189] Yunlong Lyu, Yuxuan Xie, Peng Chen, and Hao Chen. 2024. Prompt Fuzzing for Fuzz Driver Generation. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (Salt Lake City, UT, USA) *(CCS '24)*. Association for Computing Machinery, New York, NY, USA, 3793–3807. https://doi.org/10.1145/3658644.3670396

[190] Wentao Ma, Yiming Cui, Chenglei Si, Ting Liu, Shijin Wang, and Guoping Hu. 2020. CharBERT: Character-aware Pre-trained Language Model. In *Proceedings of the 28th International Conference on Computational Linguistics*. International Committee on Computational Linguistics. https://doi.org/10.18653/v1/2020.coling-main.4

[191] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-Refine: Iterative Refinement with Self-Feedback. arXiv:2303.17651 [cs.CL]

[192] Lakshmi Likhitha Mankali, Jitendra Bhandari, Manaar Alam, Ramesh Karri, Michail Maniatakos, Ozgur Sinanoglu, and Johann Knechtel. 2025. RTL-Breaker: Assessing the Security of LLMs Against Backdoor Attacks on HDL Code Generation. In *Design, Automation & Test in Europe Conference, DATE 2025, Lyon, France, March 31 - April 2, 2025*. IEEE, 1–7. https://doi.org/10.23919/DATE64628.2025.10993260

[193] Liam Daly Manocchio, Siamak Layeghy, Wai Weng Lo, Gayan K. Kulatilleke, Mohanad Sarhan, and Marius Portmann. 2024. FlowTransformer: A transformer framework for flow-based network intrusion detection systems. *Expert Syst. Appl.* 241, C (May 2024), 15 pages. https://doi.org/10.1016/j.eswa.2023.122564

[194] John Levi Martin. 2023. The Ethico-Political Universe of ChatGPT. *Journal of Social Computing* 4, 1 (2023), 1–11. https://doi.org/10.23919/JSC.2023.0003

[195] Ruijie Meng, Martin Mirchev, Marcel Böhme, and Abhik Roychoudhury. 2024. Large language model guided protocol fuzzing. In *Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS)*.

[196] Ruijie Meng, Martin Mirchev, Marcel Böhme, and Abhik Roychoudhury. 2024. Large Language Model guided Protocol Fuzzing. In *31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/large-language-model-guided-protocol-fuzzing/

[197] Xingyu Meng, Amisha Srivastava, Ayush Arunachalam, Avik Ray, Pedro Henrique Silva, Rafail Psiakis, Yiorgos Makris, and Kanad Basu. 2023. Unlocking Hardware Security Assurance: The Potential of LLMs. arXiv:2308.11042 [cs.CR]

[198] Mark Mets, Andres Karjus, Indrek Ibrus, and Maximilian Schich. 2023. Automated stance detection in complex topics and small languages: the challenging case of immigration in polarizing news media. arXiv:2305.13047 [cs.CL]

[199] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large Language Models: A Survey. arXiv:2402.06196 [cs.CL]

[200] Fatemehsadat Mireshghallah, Archit Uniyal, Tianhao Wang, David Evans, and Taylor Berg-Kirkpatrick. 2022. An Empirical Analysis of Memorization in Fine-tuned Autoregressive Language Models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 1816–1826. https://doi.org/10.18653/v1/2022.emnlp-main.119

[201] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. arXiv:1802.09089 [cs.CR]

[202] Nicolás Montes, Gustavo Betarte, Rodrigo Martínez, and Alvaro Pardo. 2021. Web Application Attacks Detection Using Deep Learning. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, João Manuel R. S. Tavares, João Paulo Papa, and Manuel González Hidalgo (Eds.). Springer International Publishing, Cham, 227–236.

[203] Osval Antonio Montesinos López, Abelardo Montesinos López, and Jose Crossa. 2022. Overfitting, model tuning, and evaluation of prediction performance. In *Multivariate statistical machine learning methods for genomic prediction*. Springer, 109–139.

[204] Kristen Moore, Cody James Christopher, David Liebowitz, Surya Nepal, and Renee Selvey. 2022. Modelling direct messaging networks with multiple recipients for cyber deception. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. 1–19. https://doi.org/10.1109/EuroSP53844.2022.00009

[205] Stephen Moskal, Sam Laney, Erik Hemberg, and Una-May O'Reilly. 2023. LLMs Killed the Script Kiddie: How Agents Supported by Large Language Models Change the Landscape of Network Threat Testing. arXiv:2310.06936 [cs.CR]

[206] Farzad Nourmohammadzadeh Motlagh, Mehrdad Hajizadeh, Mehryar Majd, Pejman Najafi, Feng Cheng, and Christoph Meinel. 2024. Large Language Models in Cybersecurity: State-of-the-Art. arXiv:2402.00891 [cs.CR]

[207] Zahra Mousavi, Chadni Islam, Kristen Moore, Alsharif Abuadbba, and M. Ali Babar. 2024. An Investigation into Misuse of Java Security APIs by Large Language Models. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security* (Singapore, Singapore) *(ASIA CCS '24)*. Association for Computing Machinery, New York, NY, USA, 1299–1315. https://doi.org/10.1145/3634737.3661134

[208] Nour Moustafa and Jill Slay. 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *2015 Military Communications and Information Systems Conference (MilCIS)*. 1–6. https://doi.org/10.1109/MilCIS.2015.7348942

[209] Niels Mündler, Mark Niklas Müller, Jingxuan He, and Martin T. Vechev. 2024. SWT-Bench: Testing and Validating Real-World Bug-Fixes with Code Agents. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (Eds.). http://papers.nips.cc/paper_files/paper/2024/hash/94f093b41fc2666376fb1f667fe282f3-Abstract-Conference.html

[210] Madhav Nair, Rajat Sadhukhan, and Debdeep Mukhopadhyay. 2023. Generating Secure Hardware using ChatGPT Resistant to CWEs. Cryptology ePrint Archive, Paper 2023/212. https://eprint.iacr.org/2023/212 https://eprint.iacr.org/2023/212.

[211] Madhav Nair, Rajat Sadhukhan, and Debdeep Mukhopadhyay. 2023. How Hardened is Your Hardware? Guiding ChatGPT to Generate Secure Hardware Resistant to CWEs. In *Cyber Security, Cryptology, and Machine Learning*, Shlomi Dolev, Ehud Gudes, and Pascal Paillier (Eds.). Springer Nature Switzerland, Cham, 320–336.

[212] Mahmoud Nazzal, Issa Khalil, Abdallah Khreishah, and NhatHai Phan. 2024. PromSec: Prompt Optimization for Secure Generation of Functional Source Code with Large Language Models (LLMs). In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (Salt Lake City, UT, USA) *(CCS '24)*. Association for Computing Machinery, New York, NY, USA, 2266–2280. https://doi.org/10.1145/3658644.3690298

[213] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. arXiv:2203.13474 [cs.LG]

[214] Vikram Nitin, Baishakhi Ray, and Roshanak Zilouchian Moghaddam. 2025. FaultLine: Automated Proof-of-Vulnerability Generation Using LLM Agents. arXiv:2507.15241 [cs.SE] https://arxiv.org/abs/2507.15241

[215] Claudio Novelli, Federico Casolari, Philipp Hacker, Giorgio Spedicato, and Luciano Floridi. 2024. Generative AI in EU Law: Liability, Privacy, Intellectual Property, and Cybersecurity. arXiv:2401.07348 [cs.CY]

[216] Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. 2024. Is Self-Repair a Silver Bullet for Code Generation? arXiv:2306.09896 [cs.CL]

[217] OpenAI. 2022. GPT-3.5. https://platform.openai.com/docs/models/gpt-3-5.

[218] OpenAI. 2023. Fine-tuning. https://platform.openai.com/docs/guides/fine-tuning.

[219] OpenAI. 2023. GPT-4. https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo.

[220] OpenAI. 2024. Technical report of Sora. https://openai.com/research/video-generation-models-as-world-simulators.

[221] Jiao Ou, Junda Lu, Che Liu, Yihong Tang, Fuzheng Zhang, Di Zhang, and Kun Gai. 2024. DialogBench: Evaluating LLMs as Human-like Dialogue Systems. arXiv:2311.01677 [cs.CL]

[222] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. arXiv:2203.02155 [cs.CL]

[223] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying Large Language Models and Knowledge Graphs: A Roadmap. *IEEE Transactions on Knowledge and Data Engineering* (2024), 1–20. https://doi.org/10.1109/tkde.2024.3352100

[224] Sudipta Paria, Aritra Dasgupta, and Swarup Bhunia. 2023. DIVAS: An LLM-based End-to-End Framework for SoC Security Analysis and Policy-based Protection. arXiv:2308.06932 [cs.CR]

[225] Kenneth G. Paterson and Douglas Stebila. 2010. One-Time-Password-Authenticated Key Exchange. In *Information Security and Privacy*, Ron Steinfeld and Philip Hawkes (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 264–281.

[226] Rishov Paul, Md. Mohib Hossain, Mohammed Latif Siddiq, Masum Hasan, Anindya Iqbal, and Joanna C. S. Santos. 2023. Enhancing Automated Program Repair through Fine-tuning and Prompt Engineering. arXiv:2304.07840 [cs.LG]

[227] Hammond Pearce, Benjamin Tan, Baleegh Ahmad, Ramesh Karri, and Brendan Dolan-Gavitt. 2023. Examining Zero-Shot Vulnerability Repair with Large Language Models. In *2023 IEEE Symposium on Security and Privacy (SP)*. 2339–2356. https://doi.org/10.1109/SP46215.2023.10179324

[228] Hammond Pearce, Benjamin Tan, Prashanth Krishnamurthy, Farshad Khorrami, Ramesh Karri, and Brendan Dolan-Gavitt. 2022. Pop Quiz! Can a Large Language Model Help With Reverse Engineering? arXiv:2202.01142 [cs.SE]

[229] Kexin Pei, Weichen Li, Qirui Jin, Shuyang Liu, Scott Geng, Lorenzo Cavallaro, Junfeng Yang, and Suman Jana. 2024. Exploiting Code Symmetries for Learning Program Semantics. arXiv:2308.03312 [cs.LG]

[230] Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The RefinedWeb Dataset for Falcon LLM: Outperforming Curated Corpora with Web Data, and Web Data Only. arXiv:2306.01116 [cs.CL]

[231] Filippo Perrina, Francesco Marchiori, Mauro Conti, and Nino Vincenzo Verde. 2023. AGIR: Automating Cyber Threat Intelligence Reporting with Natural Language Generation. arXiv:2310.02655 [cs.CR]

[232] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. 2023. Do Users Write More Insecure Code with AI Assistants?. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23)*. ACM. https://doi.org/10.1145/3576915.3623157

[233] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and software technology* 64 (2015), 1–18.

[234] Attia Qammar, Hongmei Wang, Jianguo Ding, Abdenacer Naouri, Mahmoud Daneshmand, and Huansheng Ning. 2023. Chatbots to ChatGPT in a Cybersecurity Space: Evolution, Vulnerabilities, Attacks, Challenges, and Future Recommendations. arXiv:2306.09255 [cs.CR]

[235] Jiaxing Qi, Shaohan Huang, Zhongzhi Luan, Carol Fung, Hailong Yang, and Depei Qian. 2023. LogGPT: Exploring ChatGPT for Log-Based Anomaly Detection. arXiv:2309.01189 [cs.LG]

[236] Xiangyu Qi, Yi Zeng, Tinghao Xie, Pin-Yu Chen, Ruoxi Jia, Prateek Mittal, and Peter Henderson. 2023. Fine-tuning Aligned Language Models Compromises Safety, Even When Users Do Not Intend To! arXiv:2310.03693 [cs.CL]

[237] Xingzhi Qian, Xinran Zheng, Yiling He, Shuo Yang, and Lorenzo Cavallaro. 2025. Lamd: Context-driven android malware detection and classification with llms. In *2025 IEEE Security and Privacy Workshops (SPW)*. IEEE, 126–136.

[238] Vu Le Anh Quan, Chau Thuan Phat, Kiet Van Nguyen, Phan The Duy, and Van-Hau Pham. 2023. XGV-BERT: Leveraging Contextualized Language Model and Graph Neural Network for Efficient Software Vulnerability Detection. arXiv:2309.14677 [cs.CR]

[239] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018).

[240] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[241] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2023. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. arXiv:1910.10683 [cs.LG]

[242] Mirza Masfiqur Rahman, Imtiaz Karim, and Elisa Bertino. 2024. CellularLint: a systematic approach to identify inconsistent behavior in cellular network specifications. In *Proceedings of the 33rd USENIX Conference on Security Symposium* (Philadelphia, PA, USA) *(SEC '24)*. USENIX Association, USA, Article 292, 18 pages.

[243] Priyanka Ranade, Aritran Piplai, Sudip Mittal, Anupam Joshi, and Tim Finin. 2021. Generating Fake Cyber Threat Intelligence Using Transformer-Based Models. In *2021 International Joint Conference on Neural Networks (IJCNN)*. 1–9. https://doi.org/10.1109/IJCNN52387.2021.9534192

[244] Javier Rando, Fernando Perez-Cruz, and Briland Hitaj. 2023. PassGPT: Password Modeling and (Guided) Generation with Large Language Models. arXiv:2306.01545 [cs.CL]

[245] Matthias C Rillig, Marlene Ågerstrand, Mohan Bi, Kenneth A Gould, and Uli Sauerland. 2023. Risks and benefits of large language models for the environment. *Environmental Science & Technology* 57, 9 (2023), 3464–3466.

[246] Michael Rodler, Wenting Li, Ghassan O. Karame, and Lucas Davi. 2018. Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks. arXiv:1812.05934 [cs.CR]

[247] Sayak Saha Roy, Poojitha Thota, Krishna Vamsi Naragam, and Shirin Nilizadeh. 2024. From Chatbots to Phishbots?: Phishing Scam Generation in Commercial Large Language Models . In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 36–54. https://doi.org/10.1109/SP54263.2024.00182

[248] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code Llama: Open Foundation Models for Code. arXiv:2308.12950 [cs.CL]

[249] Ahmed R. Sadik, Antonello Ceravola, Frank Joublin, and Jibesh Patra. 2023. Analysis of ChatGPT on Source Code. arXiv:2306.00597 [cs.SE]

[250] Dipayan Saha, Shams Tarek, Katayoon Yahyaei, Sujan Kumar Saha, Jingbo Zhou, Mark Tehranipoor, and Farimah Farahmandi. 2024. LLM for SoC Security: A Paradigm Shift. *IEEE Access* 12 (2024), 155498–155521. https://doi.org/10.1109/ACCESS.2024.3427369

[251] R.S. Sandhu and P. Samarati. 1994. Access control: principle and practice. *IEEE Communications Magazine* 32, 9 (1994), 40–48. https://doi.org/10.1109/35.312842

[252] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv:1910.01108 [cs.CL]

[253] Advait Sarkar. 2024. Large Language Models Cannot Explain Themselves. arXiv:2405.04382 [cs.HC] https://arxiv.org/abs/2405.04382

[254] Rebecca Saul, Hao Wang, Koushik Sen, and David Wagner. 2025. SCGAgent: Recreating the Benefits of Reasoning Models for Secure Code Generation with Agentic Workflows. arXiv:2506.07313 [cs.CR] https://arxiv.org/abs/2506.07313

[255] Mark Scanlon, Frank Breitinger, Christopher Hargreaves, Jan-Niclas Hilgert, and John Sheppard. 2023. ChatGPT for Digital Forensic Investigation: The Good, The Bad, and The Unknown. arXiv:2307.10195 [cs.CR]

[256] Jonathon Schwartz and Hanna Kurniawati. 2019. Autonomous Penetration Testing using Reinforcement Learning. arXiv:1905.05965 [cs.CR]

[257] Siti Rahayu Selamat, Robiah Yusof, and Shahrin Sahib. 2008. Mapping process of digital forensic investigation framework. *International Journal of Computer Science and Network Security* 8, 10 (2008), 163–169.

[258] Minjae Seo, Wonwoo Choi, Myoungsung You, and Seungwon Shin. 2025. AutoPatch: Multi-Agent Framework for Patching Real-World CVE Vulnerabilities. *arXiv preprint arXiv:2505.04195* (2025).

[259] Shiwen Shan, Yintong Huo, Yuxin Su, Yichen Li, Dan Li, and Zibin Zheng. 2024. Face It Yourselves: An LLM-Based Two-Stage Strategy to Localize Configuration Errors via Logs. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) *(ISSTA 2024)*. Association for Computing Machinery, New York, NY, USA, 13–25. https://doi.org/10.1145/3650212.3652106

[260] Murray Shanahan. 2023. Talking About Large Language Models. arXiv:2212.03551 [cs.CL]

[261] Xinyu She, Yanjie Zhao, and Haoyu Wang. 2024. WaDec: Decompiling WebAssembly Using Large Language Model. In *2024 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 481–492.

[262] Zhiqiang Shen, Zechun Liu, Jie Qin, Marios Savvides, and Kwang-Ting Cheng. 2021. Partial Is Better Than All: Revisiting Fine-tuning Strategy for Few-shot Learning. arXiv:2102.03983 [cs.CV]

[263] André Silva, Sen Fang, and Martin Monperrus. 2024. RepairLLaMA: Efficient Representations and Fine-Tuned Adapters for Program Repair. arXiv:2312.15698 [cs.SE]

[264] Giuseppe Siracusano, Davide Sanvito, Roberto Gonzalez, Manikantan Srinivasan, Sivakaman Kamatchi, Wataru Takahashi, Masaru Kawakita, Takahiro Kakumaru, and Roberto Bifulco. 2023. Time for aCTIon: Automated Analysis of Cyber Threat Intelligence in the Wild. arXiv:2307.10214 [cs.CR]

[265] Dominik Sobania, Martin Briesch, Carol Hanna, and Justyna Petke. 2023. An Analysis of the Automatic Bug Fixing Performance of ChatGPT. arXiv:2301.08653 [cs.SE]

[266] Qige Song, Yongzheng Zhang, Linshu Ouyang, and Yige Chen. 2022. BinMLM: Binary Authorship Verification with Flow-aware Mixture-of-Shared Language Model. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 1023–1033. https://doi.org/10.1109/SANER53432.2022.00120

[267] Robin Staab, Mark Vero, Mislav Balunović, and Martin Vechev. 2024. Beyond Memorization: Violating Privacy Via Inference with Large Language Models. arXiv:2310.07298 [cs.AI] https://arxiv.org/abs/2310.07298

[268] Nishant Subramani, Sasha Luccioni, Jesse Dodge, and Margaret Mitchell. 2023. Detecting Personal Information in Training Corpora: an Analysis. In *Proceedings of the 3rd Workshop on Trustworthy Natural Language Processing (TrustNLP 2023)*, Anaelia Ovalle, Kai-Wei Chang, Ninareh Mehrabi, Yada Pruksachatkun, Aram Galystan, Jwala Dhamala, Apurv Verma, Trista Cao, Anoop Kumar, and Rahul Gupta (Eds.). Association for Computational Linguistics, Toronto, Canada, 208–220. https://doi.org/10.18653/v1/2023.trustnlp-1.18

[269] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2020. How to Fine-Tune BERT for Text Classification? arXiv:1905.05583 [cs.CL]

[270] Tiezhu Sun, Kevin Allix, Kisub Kim, Xin Zhou, Dongsun Kim, David Lo, Tegawendé F. Bissyandé, and Jacques Klein. 2023. DexBERT: Effective, Task-Agnostic and Fine-Grained Representation Learning of Android Bytecode. *IEEE Transactions on Software Engineering* 49, 10 (2023), 4691–4706. https://doi.org/10.1109/TSE.2023.3310874

[271] Yuqiang Sun, Daoyuan Wu, Yue Xue, Han Liu, Haijun Wang, Zhengzi Xu, Xiaofei Xie, and Yang Liu. 2024. GPTScan: Detecting Logic Vulnerabilities in Smart Contracts by Combining GPT with Program Analysis. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 166, 13 pages. https://doi.org/10.1145/3597503.3639117

[272] Zhensu Sun, Li Li, Yan Liu, Xiaoning Du, and Li Li. 2022. On the Importance of Building High-quality Training Datasets for Neural Code Search. arXiv:2202.06649 [cs.SE]

[273] Christopher Swenson. 2008. *Modern cryptanalysis: techniques for advanced code breaking*. John Wiley & Sons.

[274] Hao Tang, Keya Hu, Jin Zhou, Sicheng Zhong, Wei-Long Zheng, Xujie Si, and Kevin Ellis. 2024. Code Repair with LLMs gives an Exploration-Exploitation Tradeoff. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (Eds.). http://papers.nips.cc/paper_files/paper/2024/hash/d5c56ec4f69c9a473089b16000d3f8cd-Abstract-Conference.html

[275] Siyuan Tang, Xianghang Mi, Ying Li, XiaoFeng Wang, and Kai Chen. 2022. Clues in Tweets: Twitter-Guided Discovery and Analysis of SMS Spam. arXiv:2204.01233 [cs.CR]

[276] Xunzhu Tang, Zhenghan Chen, Kisub Kim, Haoye Tian, Saad Ezzini, and Jacques Klein. 2023. Just-in-Time Security Patch Detection – LLM At the Rescue for Data Augmentation. arXiv:2312.01241 [cs.CR]

[277] Shams Tarek, Dipayan Saha, Sujan Kumar Saha, and Farimah Farahmandi. 2025. BugWhisperer: Fine-Tuning LLMs for SoC Hardware Vulnerability Detection. arXiv:2505.22878 [cs.CR] https://arxiv.org/abs/2505.22878

[278] Sheetal Temara. 2023. Maximizing Penetration Testing Success with Effective Reconnaissance Techniques using ChatGPT. arXiv:2307.06391 [cs.CR]

[279] Chandra Thapa, Seung Ick Jang, Muhammad Ejaz Ahmed, Seyit Camtepe, Josef Pieprzyk, and Surya Nepal. 2022. Transformer-Based Language Models for Software Vulnerability Detection. In *Proceedings of the 38th Annual Computer Security Applications Conference* (<conf-loc>, <city>Austin</city>, <state>TX</state>, <country>USA</country>, </conf-loc>) *(ACSAC '22)*. Association for Computing Machinery, New York, NY, USA, 481–496. https://doi.org/10.1145/3564625.3567985

[280] Kurt Thomas, Patrick Gage Kelley, David Tao, Sarah Meiklejohn, Owen Vallis, Shunwen Tan, Blaz Bratanic, Felipe Tiengo Ferreira, Vijay Kumar Eranti, and Elie Bursztein. 2024. Supporting Human Raters with the Detection of Harmful Content using Large Language Models. *CoRR* abs/2406.12800 (2024). https://doi.org/10.48550/ARXIV.2406.12800 arXiv:2406.12800

[281] Norbert Tihanyi, Tamas Bisztray, Ridhi Jain, Mohamed Amine Ferrag, Lucas C. Cordeiro, and Vasileios Mavroeidis. 2023. The FormAI Dataset: Generative AI in Software Security Through the Lens of Formal Verification. arXiv:2307.02192 [cs.DB]

[282] M. Caner Tol and Berk Sunar. 2024. ZeroLeak: Automated Side-Channel Patching in Source Code Using LLMs. In *Computer Security – ESORICS 2024: 29th European Symposium on Research in Computer Security, Bydgoszcz, Poland, September 16–20, 2024, Proceedings, Part I* (Bydgoszcz, Poland). Springer-Verlag, Berlin, Heidelberg, 290–310. https://doi.org/10.1007/978-3-031-70879-4_15

[283] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL]

[284] Saad Ullah, Mingji Han, Saurabh Pujar, Hammond Pearce, Ayse Coskun, and Gianluca Stringhini. 2023. Can Large Language Models Identify And Reason About Security Vulnerabilities? Not Yet. arXiv:2312.12575 [cs.CR]

[285] Saad Ullah, Mingji Han, Saurabh Pujar, Hammond Pearce, Ayse Coskun, and Gianluca Stringhini. 2024. LLMs Cannot Reliably Identify and Reason About Security Vulnerabilities (Yet?): A Comprehensive Evaluation, Framework, and Benchmarks. arXiv:2312.12575 [cs.CR]

[286] Orpheas van Rooij, Marcos Antonios Charalambous, Demetris Kaizer, Michalis Papaevripides, and Elias Athanasopoulos. 2021. webFuzz: Grey-Box Fuzzing for Web Applications. In *Computer Security – ESORICS 2021*, Elisa Bertino, Haya Shulman, and Michael Waidner (Eds.). Springer International Publishing, Cham, 152–172.

[287] Neeraj Varshney, Wenlin Yao, Hongming Zhang, Jianshu Chen, and Dong Yu. 2023. A Stitch in Time Saves Nine: Detecting and Mitigating Hallucinations of LLMs by Validating Low-Confidence Generation. arXiv:2307.03987 [cs.CL] https://arxiv.org/abs/2307.03987

[288] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs.CL]

[289] Alexander Wan, Eric Wallace, Sheng Shen, and Dan Klein. 2023. Poisoning language models during instruction tuning. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) *(ICML'23)*. JMLR.org, Article 1474, 13 pages.

[290] Chengpeng Wang, Wuqi Zhang, Zian Su, Xiangzhe Xu, and Xiangyu Zhang. 2024. Sanitizing Large Language Models in Bug Detection with Data-Flow. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 3790–3805. https://doi.org/10.18653/v1/2024.findings-emnlp.217

[291] Dilin Wang, Chengyue Gong, and Qiang Liu. 2019. Improving Neural Language Modeling via Adversarial Training. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 6555–6565. https://proceedings.mlr.press/v97/wang19f.html

[292] Dawei Wang, Geng Zhou, Li Chen, Dan Li, and Yukai Miao. 2024. ProphetFuzz: Fully Automated Prediction and Fuzzing of High-Risk Option Combinations with Only Documentation via Large Language Model. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (Salt Lake City, UT, USA) *(CCS '24)*. Association for Computing Machinery, New York, NY, USA, 735–749. https://doi.org/10.1145/3658644.3690231

[293] Evan Wang, Federico Cassano, Catherine Wu, Yunfeng Bai, Will Song, Vaskar Nath, Ziwen Han, Sean Hendryx, Summer Yue, and Hugh Zhang. 2024. Planning In Natural Language Improves LLM Search For Code Generation. arXiv:2409.03733 [cs.LG] https://arxiv.org/abs/2409.03733

[294] Haijun Wang, Yurui Hu, Hao Wu, Dijun Liu, Chenyang Peng, Yin Wu, Ming Fan, and Ting Liu. 2024. Skyeye: Detecting Imminent Attacks via Analyzing Adversarial Smart Contracts. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento, CA, USA) *(ASE '24)*. Association for Computing Machinery, New York, NY, USA, 1570–1582. https://doi.org/10.1145/3691620.3695526

[295] Hanbin Wang, Zhenghao Liu, Shuo Wang, Ganqu Cui, Ning Ding, Zhiyuan Liu, and Ge Yu. 2024. INTERVENOR: Prompting the Coding Ability of Large Language Models with the Interactive Chain of Repair. In *Findings of the Association for Computational Linguistics: ACL 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 2081–2107. https://doi.org/10.18653/v1/2024.findings-acl.124

[296] Huili Wang, Zhongliang Yang, Jinshuai Yang, Cheng Chen, and Yongfeng Huang. 2023. Linguistic Steganalysis in Few-Shot Scenario. *IEEE Transactions on Information Forensics and Security* 18 (2023), 4870–4882. https://doi.org/10.1109/TIFS.2023.3298210

[297] Jincheng Wang, Le Yu, and Xiapu Luo. 2024. LLMIF: Augmented Large Language Model for Fuzzing IoT Devices. In *2024 IEEE Symposium on Security and Privacy (SP)*. 881–896. https://doi.org/10.1109/SP54263.2024.00211

[298] Tao Wang, Yushu Zhang, Shuren Qi, Ruoyu Zhao, Zhihua Xia, and Jian Weng. 2023. Security and Privacy on Generative Data in AIGC: A Survey. arXiv:2309.09435 [cs.CR]

[299] Weishi Wang, Yue Wang, Shafiq Joty, and Steven C. H. Hoi. 2023. RAP-Gen: Retrieval-Augmented Patch Generation with CodeT5 for Automatic Program Repair. arXiv:2309.06057 [cs.SE]

[300] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C. H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. arXiv:2109.00859 [cs.CL]

[301] Zhaoyang Wang, Zhiyue Liu, Xiaopeng Zheng, Qinliang Su, and Jiahai Wang. 2023. RMLM: A Flexible Defense Framework for Proactively Mitigating Word-level Adversarial Attacks. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 2757–2774.

https://doi.org/10.18653/v1/2023.acl-long.155

[302] Zhilong Wang, Lan Zhang, Chen Cao, and Peng Liu. 2023. The Effectiveness of Large Language Models (ChatGPT and CodeBERT) for Security-Oriented Code Analysis. arXiv:2307.12488 [cs.CR]

[303] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903 [cs.CL]

[304] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903 [cs.CL]

[305] Yuxiang Wei, Chunqiu Steven Xia, and Lingming Zhang. 2023. Copiloting the Copilots: Fusing Large Language Models with Completion Engines for Automated Program Repair. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23)*. ACM. https://doi.org/10.1145/3611643.3616271

[306] Zhiyuan Wei, Jing Sun, Zijian Zhang, Zhe Hou, and Zixiao Zhao. 2025. Adaptive Plan-Execute Framework for Smart Contract Security Auditing. *arXiv preprint arXiv:2505.15242* (2025).

[307] Xin-Cheng Wen, Cuiyun Gao, Shuzheng Gao, Yang Xiao, and Michael R. Lyu. 2024. SCALE: Constructing Structured Natural Language Comment Trees for Software Vulnerability Detection. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) *(ISSTA 2024)*. Association for Computing Machinery, New York, NY, USA, 235–247. https://doi.org/10.1145/3650212.3652124

[308] Magdalena Wojcieszak, Andreu Casas, Xudong Yu, Jonathan Nagler, and Joshua A. Tucker. 2022. Most users do not follow political elites on Twitter; those who do show overwhelming preferences for ideological congruity. *Science Advances* 8, 39 (2022), eabn9418. https://doi.org/10.1126/sciadv.abn9418 arXiv:https://www.science.org/doi/pdf/10.1126/sciadv.abn9418

[309] Cong Wu, Jing Chen, Ziwei Wang, Ruichao Liang, and Ruiying Du. 2024. Semantic Sleuth: Identifying Ponzi Contracts via Large Language Models. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento, CA, USA) *(ASE '24)*. Association for Computing Machinery, New York, NY, USA, 582–593. https://doi.org/10.1145/3691620.3695055

[310] Fangzhou Wu, Qingzhao Zhang, Ati Priya Bajaj, Tiffany Bao, Ning Zhang, Ruoyu "Fish" Wang, and Chaowei Xiao. 2023. Exploring the Limits of ChatGPT in Software Security Applications. arXiv:2312.05275 [cs.CR]

[311] Siwei Wu, Dabao Wang, Jianting He, Yajin Zhou, Lei Wu, Xingliang Yuan, Qinming He, and Kui Ren. 2021. DeFiRanger: Detecting Price Manipulation Attacks on DeFi Applications. arXiv:2104.15068 [cs.CR]

[312] Yi Wu, Nan Jiang, Hung Viet Pham, Thibaud Lutellier, Jordan Davis, Lin Tan, Petr Babkin, and Sameena Shah. 2023. How Effective Are Neural Networks for Fixing Security Vulnerabilities. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '23)*. ACM. https://doi.org/10.1145/3597926.3598135

[313] Yulun Wu, Ming Wen, Zeliang Yu, Xiaochen Guo, and Hai Jin. 2024. Effective Vulnerable Function Identification based on CVE Description Empowered by Large Language Models. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento, CA, USA) *(ASE '24)*. Association for Computing Machinery, New York, NY, USA, 393–405. https://doi.org/10.1145/3691620.3695013

[314] Yin Wu, Xiaofei Xie, Chenyang Peng, Dijun Liu, Hao Wu, Ming Fan, Ting Liu, and Haijun Wang. 2024. AdvSCanner: Generating Adversarial Smart Contracts to Exploit Reentrancy Vulnerabilities Using LLM and Static Analysis. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento, CA, USA) *(ASE '24)*. Association for Computing Machinery, New York, NY, USA, 1019–1031. https://doi.org/10.1145/3691620.3695482

[315] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. 2023. The Rise and Potential of Large Language Model Based Agents: A Survey. arXiv:2309.07864 [cs.AI]

[316] Chunqiu Steven Xia, Matteo Paltenghi, Jia Le Tian, Michael Pradel, and Lingming Zhang. 2024. Fuzz4All: Universal Fuzzing with Large Language Models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 126, 13 pages. https://doi.org/10.1145/3597503.3639121

[317] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. 2023. Automated program repair in the era of large pre-trained language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1482–1494.

[318] Chunqiu Steven Xia and Lingming Zhang. 2022. Less training, more repairing please: revisiting automated program repair via zero-shot learning. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (<conf-loc>, <city>Singapore</city>, <country>Singapore</country>, </conf-loc>) *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 959–971. https://doi.org/10.1145/3540250.3549101

[319] Chunqiu Steven Xia and Lingming Zhang. 2023. Conversational Automated Program Repair. arXiv:2301.13246 [cs.SE]

[320] Chunqiu Steven Xia and Lingming Zhang. 2024. Automated Program Repair via Conversation: Fixing 162 out of 337 Bugs for $0.42 Each using ChatGPT. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) *(ISSTA 2024)*. Association for Computing Machinery, New York, NY, USA, 819–831. https://doi.org/10.1145/3650212.3680323

[321] Yongqin Xian, Christoph H. Lampert, Bernt Schiele, and Zeynep Akata. 2020. Zero-Shot Learning – A Comprehensive Evaluation of the Good, the Bad and the Ugly. arXiv:1707.00600 [cs.CV]

[322] Hanxiang Xu, Wei Ma, Ting Zhou, Yanjie Zhao, Kai Chen, Qiang Hu, Yang Liu, and Haoyu Wang. 2024. CKGFuzzer: LLM-Based Fuzz Driver Generation Enhanced By Code Knowledge Graph. arXiv:2411.11532 [cs.SE] https://arxiv.org/abs/2411.11532

[323] Hanxiang Xu, Yanjie Zhao, and Haoyu Wang. 2025. Directed Greybox Fuzzing via Large Language Model. arXiv:2505.03425 [cs.CR] https://arxiv.org/abs/2505.03425

[324] Minrui Xu, Jiani Fan, Xinyu Huang, Conghao Zhou, Jiawen Kang, Dusit Niyato, Shiwen Mao, Zhu Han, Xuemin, Shen, and Kwok-Yan Lam. 2025. Forewarned is Forearmed: A Survey on Large Language Model-based Agents in Autonomous Cyberattacks. arXiv:2505.12786 [cs.NI] https://arxiv.org/abs/2505.12786

[325] Shangzhi Xu, Jialiang Dong, Weiting Cai, Juanru Li, Arash Shaghaghi, Nan Sun, and Siqi Ma. 2024. Enhancing Security in Third-Party Library Reuse – Comprehensive Detection of 1-day Vulnerability through Code Patch Analysis. arXiv:2411.19648 [cs.SE] https://arxiv.org/abs/2411.19648

[326] Xiangzhe Xu, Zhuo Zhang, Shiwei Feng, Yapeng Ye, Zian Su, Nan Jiang, Siyuan Cheng, Lin Tan, and Xiangyu Zhang. 2023. LmPa: Improving Decompilation by Synergy of Large Language Model and Program Analysis. arXiv:2306.02546 [cs.SE]

[327] Xiangzhe Xu, Zhuo Zhang, Zian Su, Ziyang Huang, Shiwei Feng, Yapeng Ye, Nan Jiang, Danning Xie, Siyuan Cheng, Lin Tan, and Xiangyu Zhang. 2025. Unleashing the Power of Generative Model in Recovering Variable Names from Stripped Binary. In *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/unleashing-the-power-of-generative-model-in-recovering-variable-names-from-stripped-binary/

[328] Aidan Z. H. Yang, Ruben Martins, Claire Le Goues, and Vincent J. Hellendoorn. 2023. Large Language Models for Test-Free Fault Localization. arXiv:2310.01726 [cs.SE]

[329] Boyang Yang, Haoye Tian, Weiguo Pian, Haoran Yu, Haitao Wang, Jacques Klein, Tegawendé F. Bissyandé, and Shunfu Jin. 2024. CREF: An LLM-Based Conversational Software Repair Framework for Programming Tutors. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) *(ISSTA 2024)*. Association for Computing Machinery, New York, NY, USA, 882–894. https://doi.org/10.1145/3650212.3680328

[330] Chenyuan Yang, Yinlin Deng, Runyu Lu, Jiayi Yao, Jiawei Liu, Reyhaneh Jabbarvand, and Lingming Zhang. 2024. WhiteFox: White-Box Compiler Fuzzing Empowered by Large Language Models. *Proc. ACM Program. Lang.* 8, OOPSLA2, Article 296 (Oct. 2024), 27 pages. https://doi.org/10.1145/3689736

[331] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Bing Yin, and Xia Hu. 2023. Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond. arXiv:2304.13712 [cs.CL]

[332] Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. 2024. A Survey on Large Language Model (LLM) Security and Privacy: The Good, The Bad, and The Ugly. *High-Confidence Computing* 4, 2 (June 2024), 100211. https://doi.org/10.1016/j.hcc.2024.100211

[333] Zihao Yi, Jiarui Ouyang, Yuwen Liu, Tianhao Liao, Zhe Xu, and Ying Shen. 2024. A Survey on Recent Advances in LLM-Based Multi-turn Dialogue Systems. arXiv:2402.18013 [cs.CL]

[334] Yagmur Yigit, William J Buchanan, Madjid G Tehrani, and Leandros Maglaras. 2024. Review of Generative AI Methods in Cybersecurity. arXiv:2403.08701 [cs.CR]

[335] Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. 2024. A Survey on Multimodal Large Language Models. arXiv:2306.13549 [cs.CV]

[336] Xin Yin, Chao Ni, Shaohua Wang, Zhenhao Li, Limin Zeng, and Xiaohu Yang. 2024. ThinkRepair: Self-Directed Automated Program Repair. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) *(ISSTA 2024)*. Association for Computing Machinery, New York, NY, USA, 1274–1286. https://doi.org/10.1145/3650212.3680359

[337] Zeliang Yu, Ming Wen, Xiaochen Guo, and Hai Jin. 2024. Maltracker: A Fine-Grained NPM Malware Tracker Copiloted by LLM-Enhanced Dataset. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) *(ISSTA 2024)*. Association for Computing Machinery, New York, NY, USA, 1759–1771. https://doi.org/10.1145/3650212.3680397

[338] Wei Yuan, Quanjun Zhang, Tieke He, Chunrong Fang, Nguyen Quoc Viet Hung, Xiaodong Hao, and Hongzhi Yin. 2022. CIRCLE: continual repair across programming languages. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis* (<conf-loc>, <city>Virtual</city>, <country>South Korea</country>, </conf-loc>) *(ISSTA 2022)*. Association for Computing Machinery, New York, NY, USA, 678–690. https://doi.org/10.1145/3533767.3534219

[339] Daochen Zha, Zaid Pervaiz Bhat, Kwei-Herng Lai, Fan Yang, Zhimeng Jiang, Shaochen Zhong, and Xia Hu. 2023. Data-centric Artificial Intelligence: A Survey. arXiv:2303.10158 [cs.LG]

[340] Xian Zhan, Tianming Liu, Lingling Fan, Li Li, Sen Chen, Xiapu Luo, and Yang Liu. 2021. Research on third-party libraries in android apps: A taxonomy and systematic literature review. *IEEE Transactions on Software Engineering* 48, 10 (2021), 4181–4213.

[341] Brian Zhang and Zhuo Zhang. 2024. Detecting Bugs with Substantial Monetary Consequences by LLM and Rule-based Reasoning. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (Eds.). http://papers.nips.cc/paper_files/paper/2024/hash/f1d8400fec75f683c4d823f5836a81bb-Abstract-Conference.html

[342] Chenyuan Zhang, Hao Liu, Jiutian Zeng, Kejing Yang, Yuhong Li, and Hui Li. 2024. Prompt-Enhanced Software Vulnerability Detection Using ChatGPT. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings* (Lisbon, Portugal) *(ICSE-Companion '24)*. Association for Computing Machinery, New York, NY, USA, 276–277. https://doi.org/10.1145/3639478.3643065

[343] Cen Zhang, Yaowen Zheng, Mingqiang Bai, Yeting Li, Wei Ma, Xiaofei Xie, Yuekang Li, Limin Sun, and Yang Liu. 2024. How Effective Are They? Exploring Large Language Model Based Fuzz Driver Generation. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) *(ISSTA 2024)*. Association for Computing Machinery, New York, NY, USA, 1223–1235. https:

//doi.org/10.1145/3650212.3680355

[344] Cen Zhang, Yaowen Zheng, Mingqiang Bai, Yeting Li, Wei Ma, Xiaofei Xie, Yuekang Li, Limin Sun, and Yang Liu. 2024. How Effective Are They? Exploring Large Language Model Based Fuzz Driver Generation. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) *(ISSTA 2024)*. Association for Computing Machinery, New York, NY, USA, 1223–1235. https://doi.org/10.1145/3650212.3680355

[345] He Zhang, Muhammad Ali Babar, and Paolo Tell. 2011. Identifying relevant studies in software engineering. *Information and Software Technology* 53, 6 (2011), 625–637.

[346] Jian Zhang, Chong Wang, Anran Li, Wenhan Wang, Tianlin Li, and Yang Liu. 2024. VulAdvisor: Natural Language Suggestion Generation for Software Vulnerability Repair. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento, CA, USA) *(ASE '24)*. Association for Computing Machinery, New York, NY, USA, 1932–1944. https://doi.org/10.1145/3691620.3695555

[347] Quanjun Zhang, Chunrong Fang, Bowen Yu, Weisong Sun, Tongke Zhang, and Zhenyu Chen. 2023. Pre-Trained Model-Based Automated Software Vulnerability Repair: How Far are We? *IEEE Transactions on Dependable and Secure Computing* (2023), 1–18. https://doi.org/10.1109/TDSC.2023.3308897

[348] Quanjun Zhang, Chunrong Fang, Tongke Zhang, Bowen Yu, Weisong Sun, and Zhenyu Chen. 2023. GAMMA: Revisiting Template-based Automated Program Repair via Mask Prediction. arXiv:2309.09308 [cs.SE]

[349] Quanjun Zhang, Tongke Zhang, Juan Zhai, Chunrong Fang, Bowen Yu, Weisong Sun, and Zhenyu Chen. 2023. A Critical Review of Large Language Model on Software Engineering: An Example from ChatGPT and Automated Program Repair. arXiv:2310.08879 [cs.SE]

[350] Yuntong Zhang, Xiang Gao, Gregory J Duck, and Abhik Roychoudhury. 2022. Program vulnerability repair via inductive inference. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. 691–702.

[351] Yuwei Zhang, Zhi Jin, Ying Xing, and Ge Li. 2023. STEAM: Simulating the InTeractive BEhavior of ProgrAMmers for Automatic Bug Fixing. arXiv:2308.14460 [cs.SE]

[352] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, Longyue Wang, Anh Tuan Luu, Wei Bi, Freda Shi, and Shuming Shi. 2023. Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models. arXiv:2309.01219 [cs.CL]

[353] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024. AutoCodeRover: Autonomous Program Improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) *(ISSTA 2024)*. Association for Computing Machinery, New York, NY, USA, 1592–1604. https://doi.org/10.1145/3650212.3680384

[354] Yuwei Zhang, Ying Xing, Ge Li, and Zhi Jin. 2023. Automated Static Warning Identification via Path-based Semantic Representation. *arXiv preprint arXiv:2306.15568* (2023).

[355] Ziyin Zhang, Chaoyu Chen, Bingchang Liu, Cong Liao, Zi Gong, Hang Yu, Jianguo Li, and Rui Wang. 2024. Unifying the Perspectives of NLP and Software Engineering: A Survey on Language Models for Code. arXiv:2311.07989 [cs.CL]

[356] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A Survey of Large Language Models. arXiv:2303.18223 [cs.CL]

[357] Yuze Zhao, Zhenya Huang, Yixiao Ma, Rui Li, Kai Zhang, Hao Jiang, Qi Liu, Linbo Zhu, and Yu Su. 2024. RePair: Automated Program Repair with Process-based Feedback. In *Findings of the Association for Computational Linguistics: ACL 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 16415–16429. https://doi.org/10.18653/v1/2024.findings-acl.973

[358] Mingwei Zheng, Chengpeng Wang, Xuwei Liu, Jinyao Guo, Shiwei Feng, and Xiangyu Zhang. 2025. An LLM Agent for Functional Bug Detection in Network Protocols. arXiv:2506.00714 [cs.SE] https://arxiv.org/abs/2506.00714

[359] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, and Yongqiang Ma. 2024. LlamaFactory: Unified Efficient Fine-Tuning of 100+ Language Models. arXiv:2403.13372 [cs.CL]

[360] Zibin Zheng, Kaiwen Ning, Yanlin Wang, Jingwen Zhang, Dewu Zheng, Mingxi Ye, and Jiachi Chen. 2024. A Survey of Large Language Models for Code: Evolution, Benchmarking, and Future Trends. arXiv:2311.10372 [cs.SE]

[361] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Weili Chen, Xiangping Chen, Jian Weng, and Muhammad Imran. 2020. An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems* 105 (April 2020), 475–491. https://doi.org/10.1016/j.future.2019.12.019

[362] Li Zhong, Zilong Wang, and Jingbo Shang. 2024. Debug like a Human: A Large Language Model Debugger via Verifying Runtime Execution Step by Step. In *Findings of the Association for Computational Linguistics: ACL 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 851–870. https://doi.org/10.18653/v1/2024.findings-acl.49

[363] Shide Zhou, Tianlin Li, Kailong Wang, Yihao Huang, Ling Shi, Yang Liu, and Haoyu Wang. 2025. Understanding the Effectiveness of Coverage Criteria for Large Language Models: A Special Angle from Jailbreak Attacks. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*. IEEE, 730–742. https://doi.org/10.1109/ICSE55347.2025.00209

[364] Xin Zhou, Sicong Cao, Xiaobing Sun, and David Lo. 2024. Large Language Model for Vulnerability Detection and Repair: Literature Review and Roadmap. *arXiv preprint arXiv:2404.02525* (2024).

[365] Yuxuan Zhu, Antony Kellermann, Akul Gupta, Philip Li, Richard Fang, Rohan Bindu, and Daniel Kang. 2024. Teams of llm agents can exploit zero-day vulnerabilities. *arXiv preprint arXiv:2406.01637* (2024).