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 30K relevant papers and systematically analyzing 127 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 a wide range of cybersecurity tasks, including vulnerability detection, malware analysis, network intrusion detection, and phishing detection. Second, we find that the datasets used for training and evaluating LLMs in these tasks are often limited in size and diversity, highlighting the need for more comprehensive and representative datasets. Third, we identify several promising techniques for adapting LLMs to specific cybersecurity domains, such as fine-tuning, transfer learning, and domain-specific pre-training. Finally, we discuss the main challenges and opportunities for future research in LLM4Security, including the need for more interpretable and explainable models, the importance of addressing data privacy and security concerns, and the potential for leveraging LLMs for proactive defense and threat hunting.

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 [206] and training on massive amounts of textual data, LLMs like BERT [50], GPT-3,4 [148, 150], PaLM [41], Claude [16] and Chinchilla [79]

*Corresponding authors

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.

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

Reference	Year	Scope of topics	Dimensions of discourse	Time frame	Papers
Motlagh et al. [80]	2024	Security application	Task	2022-2023	Not specified
Divakaran et al [51]	2024	Security application	Task	2020-2024	Not specified
Yao et al. [230]	2023	Security application Security of LLM	Model Task	2019-2024	281
Yigit et al. [232]	2024	Security application Security of LLM	Task	2020-2024	Not specified
Coelho et al. [43]	2024	Security application	Task Domain specific technique	2021-2023	19
Novelli et al. [146]	2024	Security application Security of LLM	Task	2020-2024	Not specified
LLM4Security	2024	Security application	**Model** **Task** **Domain specific technique** **Data**	2020-2024	127

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 [152, 231], 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 [20, 36, 138]. 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 [30, 40, 65]. 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 [75, 135]. 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 [10, 11, 140]. 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 [93, 123]. 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 [90, 172]. 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 [31, 179, 198]. 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 [43, 51, 141, 230], 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 [146, 232], as shown in Table 1. For example, Divakaran et al. [51] 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.

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 38,112 relevant papers and systematically analyzing 127 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 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. We believe our survey can serve as a valuable resource for researchers working at the intersection of NLP, AI, and cybersecurity. The contributions of this work are summarized as follows:

- We conduct a comprehensive Systematic Literature Review (SLR) to investigate the latest research on LLM4Security, providing a mapping of the current landscape. Our search covers an extensive number of over 38,112 papers. With further quality-based and relevance-based filtering, we retain 127 papers for later detailed review.
- We formulate four key RQs to understand various aspects of LLM application in security in each distinct dimension, including the types of LLMs used, security tasks facilitated, domain specification techniques, and differences in data collection and pre-processing.
- We analyze the distribution of the 127 selected papers across venues and over time, revealing rapid growth in LLM4Security research especially in 2022-2023, and categorizes the characteristics of mainstream LLMs employed in the security domain.

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 4 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.

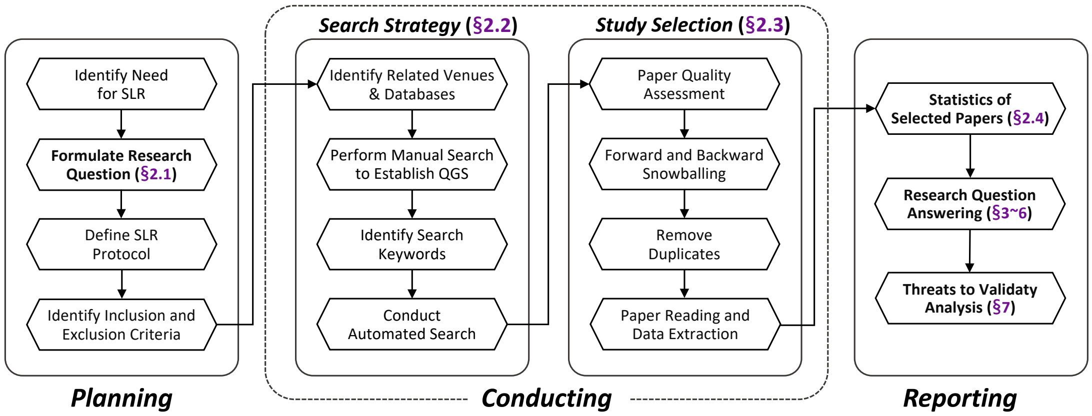


Fig. 1. Systematic Literature Review Methodology for LLM4Security.

Following the established SLR guidelines [99, 164], our methodology is structured into three pivotal stages as shown in Figure 2: 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) [239] strategy for literature search. The overview of the strategy we applied in this work is as follows:

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 [80, 236, 252].

Step2: Establish QGS. In this step, we start with creating a manually curated set of studies that have been carefully screened to form the QGS. A total of 41 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:

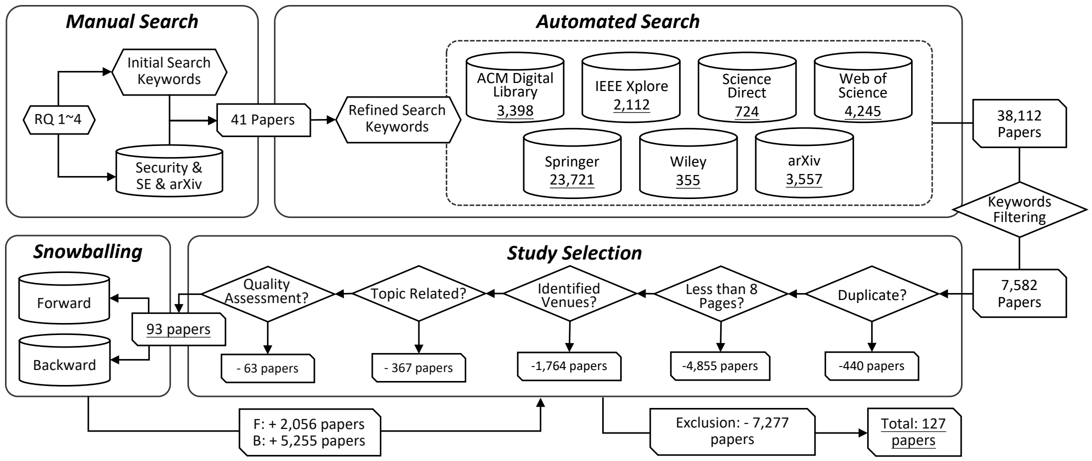


Fig. 2. Paper Search and Selection Process.

- *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.*

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 (2019-2023) is as follows: 3,398 papers in ACM Digital Library, 2,112 papers in IEEE Xplore, 724 papers in Science Direct, 4,245 papers in Web of Science, 23,721 papers in Springer, 7,154 papers in Wiley, and 3,557 papers in arXiv.

2.3 Study Selection

After obtaining the initial pool of 38,112 papers (38,071 from the automated search and 41 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 7,582 papers whose titles and abstracts contained a pair of the identified search keywords. Subsequently, Ex#1 filtered out 440

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.

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 4,855 papers being removed. The remaining papers were then screened based on the criteria Ex#3, which retained 523 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 523 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, 156 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.

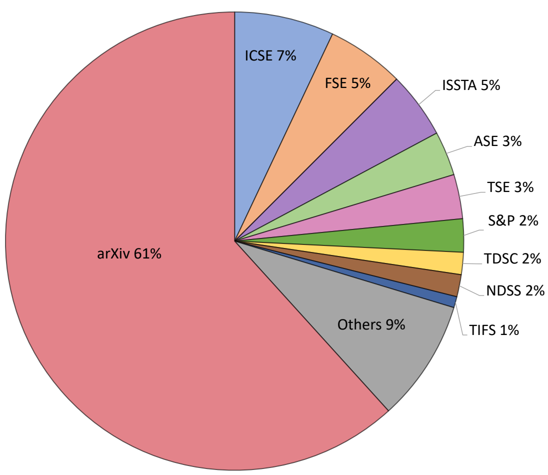
Each criterion was scored on a 3-point scale (0: not met, 1: partially met, 2: fully met). Papers with a total score of 6 or higher (out of 10) were considered as having acceptable quality. After the quality assessment, 93 papers remained in the selected set.

2.3.3 Forward and Backward Snowballing. To further expand the coverage of relevant literature, we performed forward and backward snowballing on the 93 selected papers. Forward snowballing identified papers that cited the selected papers, while backward snowballing identified papers that were referenced by the selected papers.

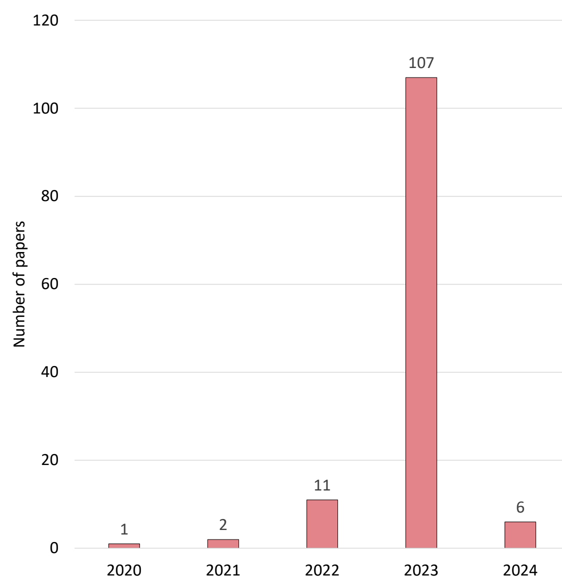
Here we obtained 2,056 and 5,255 papers separately during the forward and backward process. Then we applied the same inclusion/exclusion criteria and quality assessment to the papers found through snowballing. After the initial keyword filtering and deduplication, there were 1,978 papers that remained available. Among them, 68 papers were excluded during the page number filtering step, and 1,235 papers were deleted to ensure the papers were published in the selected venues. After confirming the paper topics and assessing the paper quality, only 44 papers were ultimately retained in the snowballing process, resulting in a final set of 127 papers for data extraction and synthesis.

2.4 Statistics of Selected Papers

After conducting searches and snowballing, a total of 127 relevant research papers were ultimately obtained. The distribution of the included documents is outlined in Figure 3. As depicted in Figure 3(A), 39% of the papers underwent peer review before publication. Among these venues, ICSE had the highest frequency, contributing 7%. Other venues making significant contributions included FSE, ISSTA, ASE, and TSE, with contributions of 5%, 5%, 3%, and 3% respectively. Meanwhile, the remaining 61% of the papers were published on arXiv, an open-access platform serving as a repository for scholarly articles. This discovery is unsurprising given the rapid emergence of new LLM4Security studies, with many works recently completed and potentially undergoing peer review. Despite lacking peer review, 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.



(A) Distribution of papers across venues.



(B) Distribution of papers over years.

Fig. 3. Overview of the selected 127 papers' distribution.

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. However, the number of papers sharply increased to 11 by 2022. Surprisingly, in 2023, the total count surged to 109 published papers. This rapid growth trend signifies an increasing interest in LLM4Security research. Currently, many works from 2024 are still under review or unpublished. Hence, we have chosen only 6 representative papers. We will continue to observe the developments in LLM4Security research throughout 2024.

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

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 127 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.

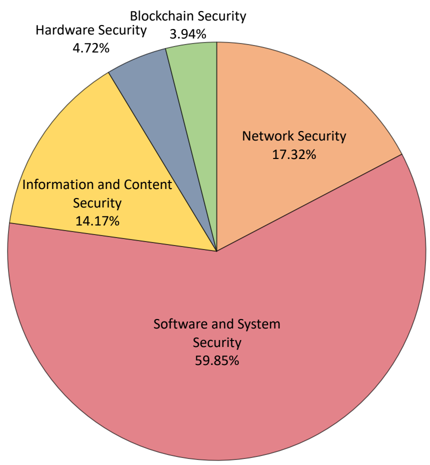


Fig. 4. Distribution of LLM usages in security domains.

The majority of research activity in the realm of software and system security, constituting around 59.84% of the total research output, is attributed to the advancements made by code LLMs [178, 247, 250] and the extensive applications of LLMs in software engineering [80]. 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 17.32% 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 14.17% 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 4.72% and 3.94% 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.

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)	22
Software and System Security	Vulnerability detection (17) Vulnerability repair (10) Bug detection (8) Bug repair (20) Program fuzzing (6) Reverse engineering and binary analysis (7) Malware detection (2) System log analysis (6)	76
Information and Content Security	Phishing and scam detection (8) Harmful contents detection (6) Steganography (2) Access control (1) Forensics (1)	18
Hardware Security	Hardware vulnerability detection (2) Hardware vulnerability repair (4)	6
Blockchain Security	Smart contract security (4) Transaction anomaly detection (1)	5

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 [205]. 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. [115] proposed GPTFuzzer based on an encoder-decoder 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 [112] fine-tuning and KL-divergence penalty to effectively generate attack payloads and mitigate the local optimum issue. Similarly, Liu et al. [120] 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 [133], 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 [137]. LLMs have been widely applied in network intrusion detection tasks, covering traditional web applications, IoT (Internet of Things), and in-vehicle network scenarios [11, 62, 131, 138]. LLMs not only learn the characteristics of malicious traffic data [10, 11, 138] and capture anomalies in user-initiated behaviors [24] but also describe the intent of intrusions and abnormal behaviors [3, 10, 58]. Additionally, they can provide corresponding security recommendations and response strategies for identified attack types [37]. Liu et al. [123] 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 [34]. 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 [189], and generating CTI from network security entity graphs [162]. Aghaei et al.'s CVEDrill [4] can generate priority recommendation reports for potential cybersecurity threats and predict their impact. Additionally, Moskal et al. [140] 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 [183]. The general penetration testing process consists of three steps: information gathering, payload construction, and vulnerability exploitation. Temara [198] 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. [31] 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. [74] developed an automated Linux privilege escalation guidance tool using LLMs. Additionally, the automated penetration testing tool PentestGPT [45], 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).

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 [216]. 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) [14]. 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 [17, 36, 38, 40, 61, 98, 124, 168, 199, 203, 211, 238, 246]. The potential demonstrated by GPT series models in vulnerability detection tasks is particularly evident [38, 61, 98, 124, 204, 238]. 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 [203]. Liu et al. [121] 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. [200] 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 [243]. Research shows that 50% of vulnerabilities have a lifecycle exceeding 438 days [110]. Delayed vulnerability patching may result in ongoing attacks on software systems [118], causing economic losses to users. The T5 model based on the encoder-decoder architecture performs better in vulnerability repair tasks [65, 240]. Although LLMs can effectively generate fixes, challenges remain in maintaining the functionality correctness of functions [158], and they are susceptible to influences from different programming languages. For example, the current capabilities of LLMs in repairing Java vulnerabilities are limited [218]. Constructing a comprehensive vulnerability repair dataset and fine-tuning LLMs on it can significantly improve the model's performance in vulnerability repair tasks [65]. Alrashedy et al. [30] proposed an automated vulnerability repair tool driven by feedback from static analysis tools. Tol et al. [201] proposed a method called ZeroLeak, which utilizes LLMs to repair side-channel vulnerabilities in programs. Charalambous et al. [12] 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.

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 [92, 111]. 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 [106, 227]. Additionally, Du et al. [54] and Li et al. [114] introduced the concept of contrastive learning, which focuses LLMs on the subtle differences between correct and buggy versions of code lines. Fang et al. [57] 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 [163]. 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 [88, 105, 222, 241], CodeT5 [88, 197, 209], Codex [56, 92, 223], LLaMa [197], CodeLLaMa [147, 188] , CodeGEN [223], UniXcoder [241], T5 [234], PLBART [88], and GPT Series [147, 197, 223, 224, 241, 242, 244] have showcased effectiveness in generating syntactically accurate and contextually relevant code. This includes frameworks with encoder-decoder architecture like Repilot [214], 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 [224]. 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 [223]. As demonstrated in the case of ChatGPT, notably enhances the accuracy of program repairs when integrated with interactive feedback loops [223]. This iterative process of patch generation and validation fosters a nuanced comprehension of software semantics, thereby resulting in more impactful fixes. 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 [65, 92, 197, 221, 240] 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 [22]. The generation capability of LLMs enables testing against various input program languages and different features [46, 220], effectively overcoming the limitations of traditional fuzz testing methods. Under strategies such as repetitive querying, example querying, and iterative querying [237], LLMs can significantly enhance the generation effectiveness of test cases. LLMs can generate test cases that trigger vulnerabilities from historical bug reports of programs [47], produce test cases similar but different from sample inputs [85], analyze compiler source code to generate programs that trigger specific optimizations [228], and split the testing requirements and test case generation using a dual-model interaction framework, assigning them to different LLMs for processing.

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 [159]. For example, Xu et al. [226] achieved recovery of variable names from binary files by propagating LLMs query results through multiple rounds. Armengol-Estapé et al. [15] 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. [193] proposed DexBert for characterizing Android system binary bytecode. Pei et al. [160] 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. [191] 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 [86] apply LLMs to enhance the readability and usability of decompiler outputs, 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 [20]. LLMs can extract semantic features of malware, leading to more competitive performance. AVScan2Vec, proposed by Joyce et al. [93], 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. Botacin [23] 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 [89]. Nowadays, researchers are utilizing the language understanding capabilities of LLMs to identify and analyze anomalies in log data. Compared to traditional deep learning methods, LLMs demonstrate outstanding performance and good interpretability [166, 185]. Fine-tuning LLMs for specific types of logs [97] or using reinforcement learning-based fine-tuning strategies [72] can significantly enhance their performance in log analysis tasks. LLMs are also being employed for log analysis in cloud servers [39, 119], where their reasoning abilities can be combined with server logs to infer the root causes of cloud service incidents.

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 [196]. LLMs can generate deceptive or false information on a large scale under specific prompts [172], making them useful for automated phishing email generation[77, 176], but compared to manual design methods, phishing emails generated by LLMs have lower click-through rates [77]. LLMs can achieve phishing email detection through prompts based on website information [100] or fine-tuning for specific email features [139, 176]. Spam emails often contain a large number of phishing emails. Labonne et al.'s research [102] has demonstrated the effectiveness of LLMs in spam email detection, showing significant advantages over traditional machine learning methods. An interesting study [28] 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 [215]. The application of LLMs in detecting harmful content can be divided into three aspects: detection of extreme political stances [73, 135], tracking of criminal activity discourse [83], and identification of social media bots [27]. 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 [75]. Hartvigsen et al. [132] 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 [13], 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. [207] 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. [18] 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 [180], 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 [156]. 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 [173].

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 [184]. Scanlon and colleagues [182] 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 [49]. 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. [134] 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. [155] 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 [179]. 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 [116]. In their study, Nair and colleagues [144] 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 Common Weakness Enumerations (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.

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 security. With the advancement of blockchain technology, smart contracts have emerged as a pivotal element in blockchain applications [251]. 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 [32, 42]. Performance varies across different vulnerability types and is constrained by the contextual length of LLMs [32]. GPTLENS [87] 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 [194] 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.

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 [175, 217]. 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. [66] 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.

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 21 cybersecurity tasks and found that LLMs are most widely applied in the field of software and system security, with 76 papers covering 8 tasks. Only 5 papers mentioned the least applied domain—blockchain security.

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 [101, 136, 186, 212, 248]. Researchers have noted substantial improvements in their performance as model size increases, with surpassing certain parameter thresholds leading to significant performance gains [79, 186]. The term "Large Language Model" (LLM) distinguishes language models based on the size of their parameters, specifically referring to large-sized PLMs [136, 248].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 [96]. In this study, we adopt to the LLM categorization framework introduced by Panet et al. [154], 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 models, as their name implies, comprise solely an encoder network. Initially designed for language understanding tasks like text classification, these models, such as BERT and its variants [5, 50, 60, 71, 76, 127, 129, 181], aim to predict a class label for input text [50]. For instance, BERT, which adopts the encoder architecture of the Transformer model, is mentioned in 35 papers included in this study. Encoder-only LLMs use a bidirectional multi-layer self-attention mechanism to calculate the relevance of each token with all other tokens, thereby capturing semantic features that include the global context. This architecture is mainly used for processing input data, focusing on understanding and encoding information rather than generating new text. Researchers employed these models to generate embeddings for data that is relavent to cybersecurity (such as traffic data and code), mapping complex data types into vector space. These models typically use a masking strategy during pre-training, and the complex training strategies increase training time and the risk of overfitting. In the realm of cybersecurity, researchers have adopted advanced models that offer capabilities much needed in cybersecurity tasks such as code understanding [211] and traffic analysis [3].

Various prominent models, including CodeBERT [60], GraphCodeBERT [71], RoBERTa [127], CharBERT [129], DeBERTa [76], and DistilBERT [181], have gained widespread usage due to their ability to effectively process and analyze code, making them valuable tools in the field of cybersecurity. An example is RoBERTa [127], which enhances BERT’s robustness through various model design adjustments and training techniques. These include altering key hyperparameters, eliminating the next-sentence pre-training objective, and utilizing substantially larger mini-batches and learning rates during training. CodeBERT [60] is a bimodal extension of BERT that utilizes both natural language and source code as its input. It employs a replaced token detection task to bolster its understanding of programming languages, in order to tackle code generation and vulnerability detection tasks. The encoder-only architecture provides models with excellent data representation capabilities. Note that 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 in NLP tasks for code semantics interpretation and understanding. In contrast, SecureBERT [5] is a BERT variant specifically designed for cyber threat analysis tasks. Its development highlights the robustness and flexibility of encoder-only architecture models across different tasks. Diverse training tasks and specialized training schemes enhance the model’s feature representation capabilities and boosts its performance in cybersecurity-related tasks.

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 (8)	2018.10	Yes
	RoBERTa (12)	2019.07	Yes
	DistilBERT (3)	2019.10	Yes
	CodeBERT (8)	2020.02	Yes
	DeBERTa (1)	2020.06	Yes
	GraphCodeBERT (1)	2020.09	Yes
	CharBERT (1)	2020.11	Yes
Encoder-Decoder	T5 (4)	2019.10	Yes
	BART (1)	2019.10	Yes
	PLBART (3)	2021.03	Yes
	CodeT5 (5)	2021.09	Yes
	UniXcoder (1)	2022.03	Yes
	Flan-T5 (1)	2022.10	Yes
Decoder-Only	GPT-2 (9)	2019.02	Yes
	GPT-3 (4)	2020.04	Yes
	GPT-Neo (1)	2021.03	Yes
	CodeX (9)	2021.07	No
	CodeGen (5)	2022.03	Yes
	InCoder (1)	2022.04	Yes
	PaLM (3)	2022.04	No
	Jurassic-1 (1)	2022.04	No
	GPT-3.5 (52)	2022.11	No
	LLaMa (4)	2023.02	Yes
	GPT-4 (38)	2023.03	No
	Bard (8)	2023.03	No
	Claude (3)	2023.03	No
	StarCoder (3)	2023.05	Yes
	Falcon (2)	2023.06	Yes
	CodeLLaMa (4)	2023.08	Yes

Regarding the model applicability, as shown in the Figure 5, encoder-only models initially garnered attention in the fields of network cybersecurity [11] and software and systems cybersecurity [106, 222]. In 2023, this concept was extended to the field of information and content cybersecurity, utilizing encoder-only models to harmful content on social media platforms [27, 73, 135].

Encoder-decoder LLMs. The Transformer model, based on the encoder-decoder architecture [206], consists of two sets of Transformer blocks: the encoder and decoder. Stacked multi-head self-attention layers are used by the encoder to encode the input sequence, generating latent representations. In contrast, the decoder performs cross-attention on these representations and sequentially produces the target sequence. The structure of encoder-decoder LLMs makes them highly suitable for sequence-to-sequence tasks such as code translation and summarization. However, their complex architecture requires more computational resources and high-quality labeled data.

Models like BART [109], T5 [171], and CodeT5 [210] exemplify this architecture. CodeT5 [210] and PLBART [9] have built upon the foundation of their original models by introducing bimodal inputs of programming language and

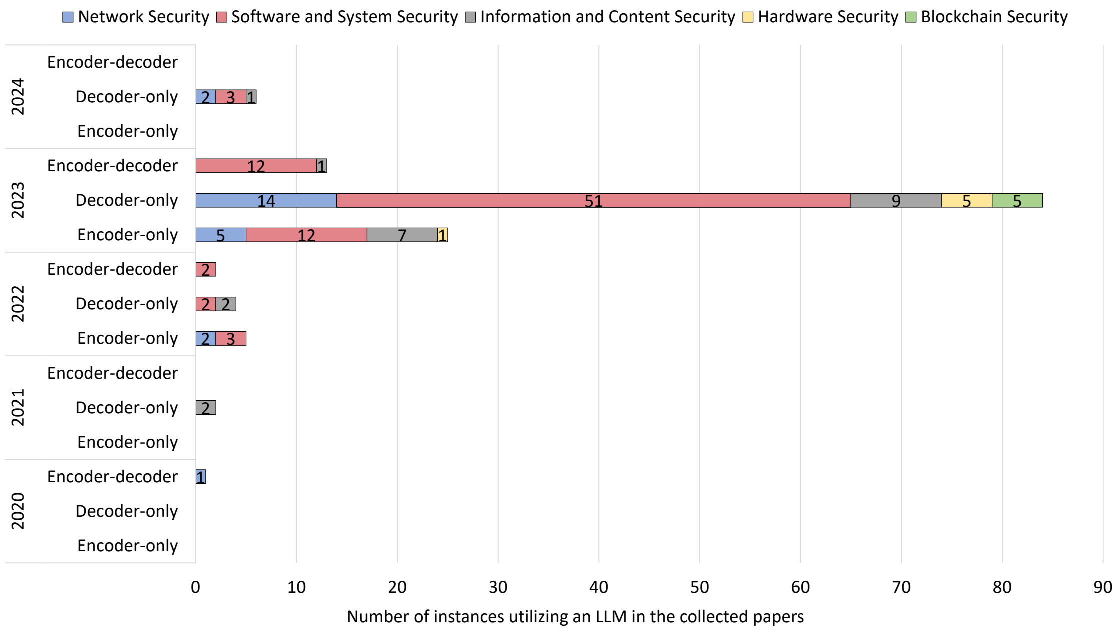


Fig. 5. Distribution and trend of different model architectures.

text, demonstrating effective code comprehension capabilities. Raffle et al. [171] show that almost all NLP tasks can be framed as a sequence-to-sequence generation task in their work. In LLM4Security, the encoder-decoder architecture was first attempted to be applied in the field of network security [120]. However, subsequent research has not widely adopted this approach, possibly due to the complexity of the encoder-decoder structure. From another perspective, owing to its flexible training strategy and excellent adaptability to complex tasks, the encoder-decoder model was later extended to other cybersecurity tasks such as program fuzzing [47], reverse engineering [15], and phishing emails detection [90].

Decoder-only LLMs. Unlike the encoder-decoder architecture, which involves the encoder processing input text and the decoder generating output text by predicting subsequent tokens from an initial state, decoder-only LLMs rely solely on the decoder module to produce the target output text [169]. This autoregressive training paradigm allows decoder-only models to generate longer-form outputs token by token, making them well-suited for producing detailed analyses, advisories, and even code relevant to cybersecurity. The attention mechanism in these models also enables them to flexibly draw upon the extensive knowledge stored in their parameters and apply it to the current context.

GPT-2 [170], GPT-3 [25], GPT-3.5 [148], and GPT-4 [150] belong to the GPT series of models, among which 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 [45, 58, 62, 182]. Their strong few-shot learning abilities allow rapid development of new cybersecurity capabilities with minimal fine-tuning. More specialized versions like Codex [33] and others have been fine-tuned for specific code-related tasks. Open-source models like GPT-Neo [21], LLaMa [202], and Falcon [161] also follow this architecture. Additionally, code generation LLMs such as CodeGen [145], InCoder [64], StarCoder [113], and CodeLLaMa [177] have been widely used for bug detection and repair, as well as for vulnerability repair [218, 224, 227].

The large context window of decoder-only models allows them to take in and utilize more context about 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 [172]. Decoder-only LLMs have gained prominence in recent years, especially in 2022 and 2023 as shown in Figure 5, witnessing a surge in development and commercial adoption by leading Internet companies. For instance, Google introduced Bard [69], while Meta unveiled LLaMa [202]. Unlike GPT-4 and its derivative application, ChatGPT, which quickly found integration into various cybersecurity tasks. These newer models have yet to see widespread adoption in the cybersecurity domain.

4.2 Trend Analysis

Illustrated in Figure 5, from 2020 to 2024, 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

Timeline and Model Architecture distribution. In 2020 and 2021, the use of LLMs in cybersecurity was limited, with only 3 research papers exploring their potential. In 2020, encoder-decoder LLMs, known for their strong performance on sequence-to-sequence tasks, were the sole architecture used in a single paper. However, in 2021, the focus shifted to decoder-only LLMs, which excel at generating longer-form outputs and handling diverse queries due to their autoregressive generation capabilities and large context windows. This shift can be attributed to the research emphasis on LLM performance in natural language processing tasks and innovations in LLM architectures during this period [25, 96].

The year 2022 marked a significant turning point, with the number of papers 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 46% of the research (5 papers). Encoder-decoder LLMs, with their strong performance on well-defined tasks, were featured in 18% (2 papers), while decoder-only LLMs, leveraging their knowledge recall and few-shot learning capabilities, garnered 36% of the research interest (4 papers). 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 significant shift towards decoder-only LLMs, which emerged as the primary architecture for addressing cybersecurity challenges. This trend is closely tied to the powerful text comprehension, reasoning capabilities [153, 213], and open-ended generation demonstrated by chatbots like ChatGPT. These decoder-only models require minimal fine-tuning and can generate both syntactically correct and functionally relevant code snippets [103, 178]. In 2023, decoder-only LLMs accounted for 68.9% of the total research, while encoder-decoder LLMs and encoder-only LLMs contributed 10.7% (14 papers) and 22.1% (27 papers), respectively. Remarkably, all studies conducted in 2024 utilized the decoder-only architecture, indicating a strong focus on exploring and leveraging the unique advantages of these models in cybersecurity research and applications.

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], a popular open-source model community, reveals that while encoder-only models like BERT and its variants have the highest number of downloads, decoder-only models are gaining significant traction. Moreover, 16 out of the top 20 most liked LLMs are decoder-only models, indicating a strong preference and excitement for their potential 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 [59].

Applying LLMs to cybersecurity. In our research, the use of LLMs can be categorized into agent-based processing and fine-tuning for specific tasks. Closed-source LLMs, represented by the GPT series, are the most popular in our studies. Researchers access LLMs online by calling APIs provided by LLM publishers and design task-specific prompts to guide LLMs to solve real-world problems with their training data [53, 91, 130], such as vulnerability repair and penetration testing [38, 45, 218]. Another approach involves locally fine-tuning open-source LLMs, by using datasets customized for specific functionalities, where researchers are able to achieve significant performance improvements [188, 227].

In summary, the transition of LLMs in cybersecurity, progressing from encoder-only architectures to decoder-only architectures, underscores the dynamic nature and flexibility of the field. This change has fundamentally altered the method for addressing cybersecurity tasks, signaling ongoing innovation within the discipline.

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 98 papers have investigated the utilization of decoder-only LLMs in cybersecurity tasks.

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

LLMs have demonstrated their efficacy across various intelligent tasks [94]. 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 [192], encompassing the adjustment of LLM parameters to suit specific tasks. This process entails training the model on task-relevant datasets, with the extent of fine-tuning contingent upon task complexity and dataset size [52, 167]. Fine-tuning can mitigate the constraints posed by model size, enabling smaller models fine-tuned for specific tasks to outperform larger models lacking fine-tuning [98, 249]. We classify fine-tuning techniques employed in papers leveraging LLMs for security tasks into two categories: full fine-tuning and partial fine-tuning. Notably, many papers employ fine-tuning without explicitly specifying the technique. In such cases, if an open-source LLM is utilized, we presume full fine-tuning; if a closed-source LLM like GPT series models is utilized, we assume partial fine-tuning.

A total of 32 papers in this study applied fine-tuning techniques to address security tasks. Among them, the most popular approach is full fine-tuning, with 23 papers, accounting for 71.88% of the total. This may be because the pre-training tasks of LLMs are far from the content of the security tasks being applied, thus requiring updating all parameters of LLMs to achieve more competitive performance. partial fine-tuning is also highly regarded, with 28.12% of the papers choosing this approach to fine-tune LLMs. As shown in Table 7, full fine-tuning has a wide range of applications, including information and content security, network security, and software and system security. The most widespread domain among them is software and system security, totaling 16 papers, accounting for 65.57% of the total. A similar distribution is also observed in partial fine-tuning, with the most widely applied domain still being software and system security, totaling 7 papers, accounting for 77.78%. The applicability of fine-tuning techniques in these security tasks indicates that pre-training LLMs may not adequately address these security tasks, and updating model parameters on specific datasets is necessary to enhance effectiveness. The choice between full fine-tuning and partial fine-tuning depends on the balance between performance and efficiency considerations.

Table 7. Distribution of fine-tuning techniques adopted in papers and the numbers in parentheses represent the number of papers.

Fine-tuning technique	Security task	Reference
Full fine-tuning	Bug detection (1)	[57]
	Access control (1)	[173]
	Steganography (1)	[207]
	Reverse engineering and binary analysis (1)	[193]
	Traffic and intrusion detection (1)	[62]
	Phishing and scam detection (2)	[172] [90]
	Harmful contents detection (2)	[73] [135]
	System log analysis (2)	[97] [72]
	Vulnerability repair (3)	[218] [65] [240]
	Bug repair (4)	[234] [157] [88] [209]
	Vulnerability detection (5)	[38] [61] [199] [246] [98]
Partial fine-tuning	Traffic and intrusion detection (1)	[10]
	Harmful contens detection (1)	[83]
	Program fuzzing (1)	[47]
	Bug repair (2)	[92] [188]
	Bug detection (2)	[106] [227]
	Vulnerability detection (2)	[36] [98]

Full fine-tuning. Full fine-tuning involves adjusting all parameters of the LLMs, including every layer of the model, to align with the specific requirements of the target task. This approach is favored when there exists a substantial disparity between the task and the pre-trained model or when the task necessitates the model to possess high adaptability and flexibility. Although full fine-tuning demands significant computational resources and time, it often yields superior performance [128]. The success of full fine-tuning relies on having a dataset tailored to the task at hand. For instance, in bug fixing tasks, a dataset containing bug-patch pairs is essential to familiarize the LLMs with the intricacies of the target task [88, 209]. LLM4Security encompasses a range of tasks, including bug repair [88, 157, 209, 234], vulnerability detection and repair [38, 65, 246], phishing, and harmful content detection [90, 135], among others, where full fine-tuning plays a crucial role in achieving optimal results.

Partial fine-tuning. Partial fine-tuning of LLMs is employed in some of the papers we collected, primarily to address security tasks while considering computational resource limitations and model copyright constraints. Partial fine-tuning involves updating only the top layers or a few layers of the model during the fine-tuning process, while keeping the lower-level parameters of the pre-trained model unchanged [187]. The aim of this approach is to retain the general knowledge of the pre-trained model while adapting to the specific task by fine-tuning the top layers. This method is typically used when there is some similarity between the target task and the LLMs, or when the task dataset is small. In LLM4Security, the partial fine-tuning techniques applied can be categorized into API fine-tuning [10, 47, 83, 92, 98, 149], adapter-tuning [81, 106, 227], prompt-tuning [36, 108], and Low-Rank Adaptation (LoRA) [84, 188]. These techniques ensure the effectiveness of LLMs in downstream security tasks while requiring smaller computational resource overhead.

5.2 Prompting LLMs for Security Tasks

Recent studies in natural language processing highlight the significance of prompt engineering [122] as an emerging fine-tuning approach aimed at bridging the gap between the output expectations of large language models 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 large language models like ChatGPT, Bard, among others. These customized prompts serve a dual purpose: they direct the large language models 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 [189], 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 [225], extracting structured network threat intelligence from unstructured data, providing standardized threat descriptions, and formalized categorization. In the context of program fuzzing tasks [85], 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 [19], even with limited input. For tasks such as penetration testing [45] and hardware vulnerability verification [179], which involve multiple steps and strict logical reasoning relationships between steps, one can utilize a chain of thought (COT) [213] 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 Table7 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 [117]. 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 8, with 7 different external augmentation techniques. The first augmentation technique we focus on is feature augmentation. The effectiveness of LLMs 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 vulnerabilitiess [242], bug locations [92], threat flow graphs [121], 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 [162]. Rule-based external tools can also serve as specialized external knowledge repositories. In addressing security tasks, LLMs can utilize results from external tools to rectify their outputs, thereby avoiding redundancy and errors[12, 74]. The fourth augmentation technique is task-adaptive training. Existing studies adopt various training strategies from pre-training to strengthen LLMs' adaptability to complex security tasks, enabling them to generate

Table 8. 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.	[92] [237] [220] [90] [10] [242] [121]
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.	[54] [209] [162]
External tools	Analysis results from specialized tools serving as auxiliary inputs.	Static code analysis tools, penetration testing tools.	[74] [12] [15]
Task-adaptive training	Different training strategies from pre-training to enhance the model's adaptability to the task.	Contrastive learning, transfer learning, reinforcement learning, distillation.	[57] [106] [240] [160] [72] [115] [197] [27]
Inter-model interaction	Introducing multiple models (which can be LLMs or other models) to collaborate and interact.	Multiple LLMs feedback collaboration, graph neural networks	[27] [228] [197]
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	[226] [234]
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	[36] [200]

more targeted outputs. For instance, contrastive learning techniques can be employed, where both bugs and patches are used as input to LLMs to automatically generate higher-quality program patches [35, 197]. Alternatively, reinforcement learning can guide LLMs to produce more effective web test cases and alleviate local optima issues [63, 115]. 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 [228]. 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 [226, 234], it is necessary to construct prompts for LLMs multiple times continuously to iterate towards the desired output. In this process, broadcasting the output results of each step iteratively as part of the prompt for the next step helps reinforce the contextual relationship between each interaction, thereby reducing error rates. The final augmentation technique is post-processing, where LLMs' outputs are validated or processed for certain security tasks requiring specific types of output [200]. This process helps mitigate issues such as hallucinations arising from the lack of domain knowledge in LLMs [36].

External augmentation techniques have significantly boosted the effectiveness of LLMs across various security tasks, yielding competitive performance. We observed that only 28 out of the total 127 papers in LLM4Security, accounting for 22.05%, applied specific external augmentation techniques. 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 127 papers employing this approach. Fine-tuning techniques were used in 25.2% of the papers, while task-specific external augmentation techniques were adopted in 22.05% 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.

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 [235]. 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 [195]. Sufficient, high-quality, and diverse data areare 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 [27, 32, 131, 238]. 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 [142]. 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 social media, or vulnerability information from CVE websites, tailored to specific research questions.

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, 100, 140, 218]. These changes could be made through manual or semi-automated processes, which might entail creating test sets tailored to

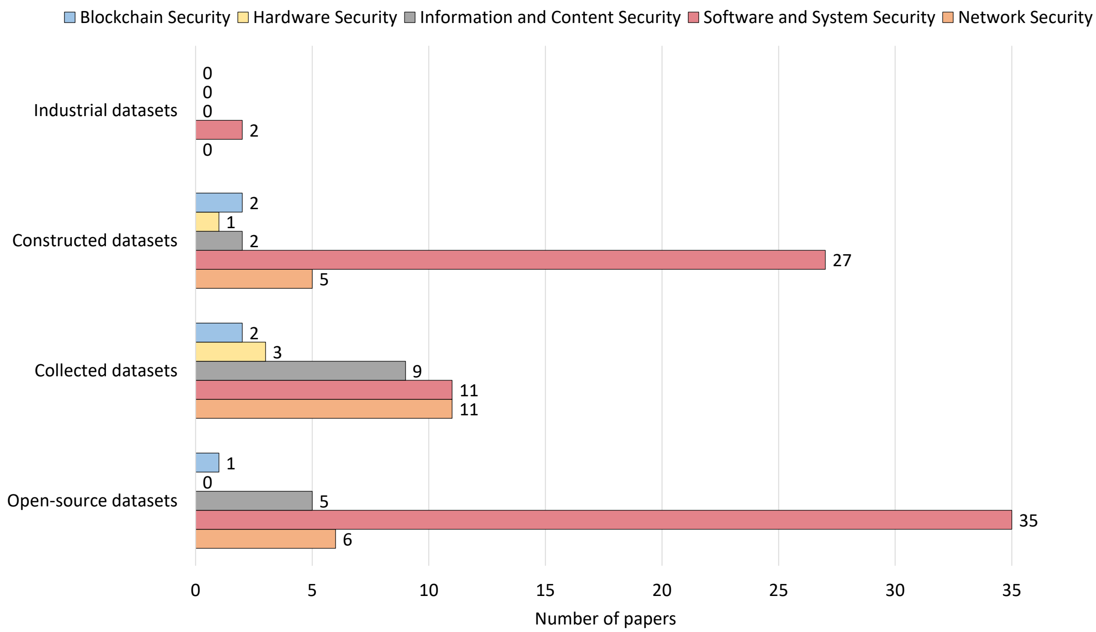


Fig. 6. The collection strategies of datasets in LLM4Security.

specific domains, annotating datasets, or generating synthetic data. For instance, researchers might gather information on web vulnerabilities and the corresponding penetration testing methods, structure them into predefined templates to form vulnerability scenarios, and subsequently assess large language models using these scenarios [45].

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 [119, 237]. These datasets are particularly valuable for research aimed at addressing real-world 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 47 studies utilize open-source datasets to train LLMs. The utilization of open-source datasets for training LLMs 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. 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 and construct datasets 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 127 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 large language models (LLMs) across academic and industrial domains. Some papers focus on exploring the use of existing LLMs, such as ChatGPT, in security tasks [143, 144]. 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.

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. This decision significantly impacts the overall performance and ability of LLMs to generalize [125]. 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 hybrid data types. Table 9 provides a detailed breakdown of the specific data included in each category, derived from 127 studies. The analysis reveals that the majority of studies rely on code-based datasets, constituting a total of 71 datasets. This dominance underscores the superior code analysis capabilities of LLMs when trained for security tasks. These models demonstrate proficiency in understanding and processing code data, making them well-suited for security challenges such as vulnerability detection, program fuzzing, and traffic analysis. Their capacity to handle and learn from extensive code data enables LLMs to offer robust insights and solutions for various security applications.

Text datasets with numerous prompts (a total of 28) are commonly utilized for tasks lacking structured data, effectively guiding large language models (LLMs) through prompts to influence their behavior. 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. This is because black-box models can be fine-tuned with small-sized data inputs during usage. Among the 127 papers analyzed, text datasets rich in prompts are frequently used for training LLMs in security tasks, highlighting this trend. Additionally, specific security tasks necessitate particular text data inputs, such as system log analysis and harmful content detection.

The prevalence of vulnerable code (17), source code (15), and bug-fix pairs (14) in code-based datasets can be attributed to their ability to effectively meet task requirements. Vulnerable code naturally exhibits semantic features of code containing vulnerabilities to large language models (LLMs), thereby highlighting the distinguishing traits of vulnerable code when juxtaposed with normal code snippets. This aids LLMs in performing security tasks related to vulnerability detection. A similar rationale applies to bug-fix pairs. Source code serves as the backbone of any 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 to grasp the intricacies of programs, enabling them to proficiently generate, analyze, and comprehend code across various security tasks. Additionally, commonly used data types for bug fixes and traffic and intrusion detection, such as bugs (7) and traffic packets (4), are also widespread.

Some studies have utilized composite datasets containing multiple data types, such as vulnerable code and vulnerability descriptions. For instance, Liu et al. [124] collected a dataset comprising CVE vulnerable code along with vulnerability descriptions and evaluated the performance of LLMs on vulnerability description mapping tasks based on this dataset.

Table 9. Data types of datasets involved in prior studies.

Category	Data type	Studies	Total	References
Code-based datasets	Vulnerable code	17	71	[38] [61] [36] [40] [124] [199] [238] [246] [98] [203] [121] [218] [30] [12] [7] [32] [204]
	Source code	15		[17] [85] [46] [228] [226] [193] [15] [160] [191] [121] [194] [42] [87] [66] [86]
	Bug-fix pairs	14		[92] [114] [244] [234] [157] [147] [222] [88] [241] [224] [223] [209] [242] [188]
	Bugs	7		[111] [106] [190] [157] [88] [47] [8]
	Traffic packages	4		[138] [62] [131] [11]
	Patches	3		[98] [105] [197]
	Code changes	3		[227] [54] [214]
	Vulnerability-fix pairs	2		[240] [65]
	Bug fixing commits	2		[244] [209]
	Web attack payloads	2		[120] [115]
	Subject protocol programs	1		[133]
	Vulnerable programs	1		[158]
Text-based datasets	Prompts	17	49	[10] [140] [45] [198] [31] [74] [56] [201] [237] [159] [23] [77] [176] [132] [182] [179] [116]
	Log messages	6		[119] [39] [97] [166] [72] [185]
	Social media contents	5		[73] [83] [27] [135] [207]
	Spam messages	4		[102] [139] [28] [90]
	Bug reports	3		[57] [106] [54]
	Attack descriptions	2		[24] [58]
	CVE reports	2		[3] [4]
	Cyber threat intelligence data	2		[172] [100]
	Top-level domains	1		[123]
	Security reports	1		[5]
	Threat reports	1		[189]
	Structured threat information	1		[162]
	Program documentations	1		[220]
	Antivirus scan reports	1		[93]
	Passwords	1		[173]
	Hardware documentations	1		[134]
Combined datasets	Vulnerable code and vulnerability descriptions	2	2	[124] [36]

Table 10. 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.	[193] [29] [138]
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.	[199] [238] [242]
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.).	[61] [98] [234] [157]
Code representation	Represented as tokens.	Tokenize source or binary code as tokens.	[246] [222] [88]
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.	[227] [191]

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 [106]. 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.

Data preprocessing techniques for code-based datasets. We outline the preprocessing techniques utilized for code-based datasets, comprising five essential steps. Table 10 provides a comprehensive summary of each technique with examples. The initial step involves extracting data, retrieving relevant code snippets from diverse sources. Depending on the research task's needs [138, 193], snippets may be extracted at different levels of detail, ranging from individual lines, methods, and functions to entire code files or projects. To prevent bias and redundancy during training, the next step removes duplicate instances by identifying and eliminating them from the dataset [238, 242], enhancing diversity and uniqueness. Filtering follows, removing snippets that don't meet predefined quality standards to ensure relevance to the security task and avoid noise [61, 234]. Code representation converts snippets into suitable formats for LLM processing, often utilizing token-based representations for security tasks [222]. Finally, data splitting divides the preprocessed dataset into training, validation, and testing subsets [227]. Training sets train the LLM, validation sets tune hyperparameters, and testing sets assess model performance on unseen data. By adhering to these steps, researchers can construct structured code-based datasets, facilitating LLM application across various security tasks like vulnerability detection, program fuzzing, and intrusion detection.

Table 11. The data preprocessing techniques for text-based datasets.

Preprocessing techniques	Description	Examples	References
Data extraction	Retrieve appropriate text from documentation based on various software engineering tasks.	Attack description, bug reports, social media content, hardware documentation, etc.	[58] [3] [220] [73] [134]
Initial data segmentation	Categorize data into distinct groups as needed.	Split data into sentences or words.	[102] [135] [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.	[57] [27] [5]
Text representation	Token-based text representation.	Tokenize the texts, sentences, or words into tokens.	[4] [134]
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.	[173] [207] [123]

Data preprocessing techniques for text-based datasets. As depicted in Table 11, 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 [57], program documentation [220], hardware documentation [134], and social media content [73]. 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, 134]. Subsequent preprocessing operations standardize and clean the text, typically involving the removal of specific symbols, stop words, and special characters [4, 135]. 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 [102]. 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.

RQ4 - Summary

(1) Based on different data sources, datasets are categorized into four types: open-source datasets, collected datasets, constructed datasets, and industrial datasets. The use of open-source datasets is the most common, accounting for approximately 38.52% in the 122 papers explicitly mentioning dataset sources. Collected datasets and constructed datasets are also popular, reflecting the lack of practical data in LLM4Security research.
(2) We categorize all datasets into three types: code-based, text-based, and combined. Text-based and code-based types are the most commonly used types when applying LLMs to security tasks. This pattern indicates that LLMs excel in leveraging their natural language processing capabilities to handle text-based and code-based data in security tasks.
(3) We summarize the data preprocessing process for different data types, outlining common data preprocessing steps such as data extraction, unqualified data deletion, data representation, and data segmentation.

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 [229]. Models with billions or even trillions of parameters present substantial challenges in terms of storage, memory, and computational demands [59]. 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 [60] emerged in 2019 as a pre-trained model featuring 125M parameters and a model size of 476MB. Recent models like Codex [33] and CodeGen [145] 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 [21] 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 [174]. 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. [82] 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 scarcity. 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 [107]. 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 [55].

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 3, 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. Recent research [163, 208] has 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 the development of technologies and tools that offer deeper insights into model internals, empowering developers to comprehend the rationale behind generated outputs. Improving interpretability and trustworthiness can ultimately foster the widespread adoption of cost-effective automation in the cybersecurity domain, fostering more efficient and effective security practices. 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 [44].

8.2 Opportunities

8.2.1 *Improvement of LLM4Security.*

Training models for security tasks. Deciding between commercially available pre-trained models like GPT-4 [150] and open-source frameworks such as T5 [171] or LLaMa [202] 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 [78]. 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 [228] or combining LLMs with specialized machine learning models [27, 197] 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 127 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 [104]. Beyond generating task solutions, ChatGPT promotes collaboration, signaling a broader effort to integrate advanced natural language understanding into traditional cybersecurity practices [45, 165]. 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 [95], a tendency to generate hallucinations [245], 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 [54, 67]. Another approach is to incorporate external specialized tools to provide real-time interactive feedback to guide LLMs [12, 15], 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 [194]. 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 [26].

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 [233]. 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, 117]. This integration aims to enhance LLMs' ability to address security tasks, especially those requiring a significant amount of domain expertise, such as penetration testing [45, 198], hardware vulnerability detection [49], log analysis [97, 119], 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.

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 [36, 115, 220]. However, the inherent lack of domain-specific knowledge and hallucinations in LLMs restrict their ability to perceive task requirements or environments with high quality [245]. 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 [219]. 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 [36, 92, 162]. 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 [151], 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 remains a pressing area for investigation [230]. The intricate nature of LLMs renders them vulnerable to attacks, necessitating innovative strategies to fortify the models themselves [44, 68, 126]. Previous studies have identified vulnerabilities in LLMs like jailbreaking and malicious prompt injection, resulting in the exposure of model training data or sensitive user chat records [48, 70, 126]. Considering that the inputs for security tasks often involve security-sensitive data (such as system logs and vulnerability code in programs) [158, 166], the leakage of such information would pose significant cybersecurity risks. An intriguing avenue for future research is to empower LLMs to autonomously detect and identify their 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 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

LLMs are making waves in the cybersecurity field, with their ability to tackle complex tasks potentially reshaping many cybersecurity practices and tools. In this comprehensive literature review, we delved into the emerging uses of LLMs in cybersecurity. We firstly explored the diverse array of security tasks where LLMs have been deployed, highlighting their practical impacts (RQ1). Our analysis covered the different LLMs employed in security tasks, discussing their unique traits and applications (RQ2). Additionally, we examined domain-specific techniques for applying LLMs to security tasks (RQ3). Lastly, we scrutinized the data collection and preprocessing procedures, underlining the importance of well-curated datasets in effectively applying LLMs to address security challenges (RQ4). We outlined key challenges facing LLM4Security and provided a roadmap for future research, outlining promising avenues for exploration.

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. 2023. Fixing Hardware Security Bugs with Large Language Models. arXiv:2302.01215 [cs.CR]

[9] 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]

[10] 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]

[11] 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

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

[13] 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.

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

[15] Jordi Armengol-Estapé, Jackson Woodruff, Chris Cummins, and Michael F. P. O'Boyle. 2024. SLaDe: A Portable Small Language Model Decompiler for Optimized Assembly. arXiv:2305.12520 [cs.PL]

[16] 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]

[17] 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]

[18] 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]

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

[20] 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]

[21] 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

[22] 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

[23] 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

[24] 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.

[25] 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]

[26] 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]

[27] 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. arXiv:2306.17408 [cs.AI]

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

[29] 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]

[30] 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]

[31] 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]

[32] Chong Chen, Jianzhong Su, Jiachi Chen, Yanlin Wang, Tingting Bi, Yanli Wang, Xingwei Lin, Ting Chen, and Zibin Zheng. 2023. When ChatGPT Meets Smart Contract Vulnerability Detection: How Far Are We? arXiv:2309.05520 [cs.SE]

[33] 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]

[34] 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.

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

[36] 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]

[37] 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]

[38] 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. arXiv:2304.00409 [cs.CR]

[39] 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]

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

[41] 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]

[42] 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]

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

[44] 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

[45] Gelei Deng, Yi Liu, Víctor Mayoral-Vilches, Peng Liu, Yuekang Li, Yuan Xu, Tianwei Zhang, Yang Liu, Martin Pinzger, and Stefan Rass. 2023. PentestGPT: An LLM-empowered Automatic Penetration Testing Tool. arXiv:2308.06782 [cs.SE]

[46] 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

[47] Yinlin Deng, Chunqiu Steven Xia, Chenyuan Yang, Shizhuo Dylan Zhang, Shujing Yang, and Lingming Zhang. 2023. Large Language Models are Edge-Case Fuzzers: Testing Deep Learning Libraries via FuzzGPT. arXiv:2304.02014 [cs.SE]

[48] Erik Derner and Kristina Batistič. 2023. Beyond the Safeguards: Exploring the Security Risks of ChatGPT. arXiv:2305.08005 [cs.CR]

[49] 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.

[50] 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]

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

[52] 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]

[53] 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]

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

[55] 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]

[56] 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

[57] 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

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

[59] 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]

[60] 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]

[61] 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]

[62] 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. arXiv:2306.14263 [cs.CR]

[63] 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

[64] 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]

[65] 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

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

[67] 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]

[68] 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]

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

[70] 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. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*. 79–90.

[71] 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]

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

[73] 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]

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

[75] 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. arXiv:2203.09509 [cs.CL]

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

[77] Fredrik Heiding, Bruce Schneier, Arun Vishwanath, Jeremy Bernstein, and Peter S. Park. 2023. Devising and Detecting Phishing: Large Language Models vs. Smaller Human Models. arXiv:2308.12287 [cs.CR]

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

[79] 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]

[80] 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]

[81] 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]

[82] 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]

[83] Chuanbo Hu, Bin Liu, Xin Li, and Yanfang Ye. 2023. Unveiling the Potential of Knowledge-Prompted ChatGPT for Enhancing Drug Trafficking Detection on Social Media. arXiv:2307.03699 [cs.CL]

[84] 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]

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

[86] 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

[87] 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]

[88] 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.

[89] 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

[90] Suhaima Jamal and Hayden Wimmer. 2023. An Improved Transformer-based Model for Detecting Phishing, Spam, and Ham: A Large Language Model Approach. arXiv:2311.04913 [cs.CL]

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

[92] Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. 2023. InferFix: End-to-End Program Repair with LLMs. arXiv:2303.07263 [cs.SE]

[93] 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]

[94] 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]

[95] 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]

[96] 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]

[97] 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]

[98] 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]

[99] 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.

[100] Takashi Koide, Naoki Fukushi, Hiroki Nakano, and Daiki Chiba. 2024. Detecting Phishing Sites Using ChatGPT. arXiv:2306.05816 [cs.CR]

[101] 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]

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

[103] 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]

[104] 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]

[105] 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

[106] 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

[107] 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]

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

[109] 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]

[110] 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

[111] 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]

[112] 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]

[113] 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]

[114] 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]

[115] 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

[116] 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]

[117] 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]

[118] 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

[119] 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]

[120] 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

[121] 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]

[122] 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]

[123] 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]

[124] 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

[125] 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]

[126] 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).

[127] 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]

[128] 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]

[129] 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

[130] 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]

[131] 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 Systems with Applications* 241 (2024). https://doi.org/10.1016/j.eswa.2023.122564

[132] 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

[133] 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)*.

[134] 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]

[135] 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]

[136] 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]

[137] 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]

[138] 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.

[139] 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

[140] 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]

[141] 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]

[142] 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

[143] 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.

[144] 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.

[145] 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]

[146] 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]

[147] 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]

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

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

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

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

[152] 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]

[153] 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]

[154] 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

[155] 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]

[156] 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.

[157] 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]

[158] 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

[159] 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]

[160] 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]

[161] 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]

[162] 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]

[163] 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

[164] 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.

[165] 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]

[166] 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]

[167] 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]

[168] 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]

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

[170] 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.

[171] 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]

[172] 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

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

[174] 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.

[175] 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]

[176] Sayak Saha Roy, Poojitha Thota, Krishna Vamsi Naragam, and Shirin Nilizadeh. 2024. From Chatbots to PhishBots? – Preventing Phishing scams created using ChatGPT, Google Bard and Claude. arXiv:2310.19181 [cs.CR]

[177] 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]

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

[179] Dipayan Saha, Shams Tarek, Katayoon Yahyaei, Sujan Kumar Saha, Jingbo Zhou, Mark Tehranipoor, and Farimah Farahmandi. 2023. LLM for SoC Security: A Paradigm Shift. arXiv:2310.06046 [cs.CR]

[180] 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

[181] 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]

[182] 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]

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

[184] 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.

[185] 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. *arXiv preprint arXiv:2404.00640* (2024).

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

[187] 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]

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

[189] 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]

[190] 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]

[191] 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

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

[193] 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

[194] Yuqiang Sun, Daoyuan Wu, Yue Xue, Han Liu, Haijun Wang, Zhengzi Xu, Xiaofei Xie, and Yang Liu. 2023. GPTScan: Detecting Logic Vulnerabilities in Smart Contracts by Combining GPT with Program Analysis. arXiv:2308.03314 [cs.CR]

[195] 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]

[196] 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]

[197] 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]

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

[199] 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

[200] 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]

[201] M. Caner Tol and Berk Sunar. 2023. ZeroLeak: Using LLMs for Scalable and Cost Effective Side-Channel Patching. arXiv:2308.13062 [cs.CR]

[202] 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]

[203] 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]

[204] 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]

[205] 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.

[206] 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]

[207] 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

[208] 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]

[209] 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]

[210] 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]

[211] 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]

[212] 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]

[213] 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]

[214] 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

[215] 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

[216] 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]

[217] 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]

[218] 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

[219] 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]

[220] Chunqiu Steven Xia, Matteo Paltenghi, Jia Le Tian, Michael Pradel, and Lingming Zhang. 2024. Fuzz4All: Universal Fuzzing with Large Language Models. arXiv:2308.04748 [cs.SE]

[221] 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.

[222] 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

[223] Chunqiu Steven Xia and Lingming Zhang. 2023. Conversational Automated Program Repair. arXiv:2301.13246 [cs.SE]
[224] Chunqiu Steven Xia and Lingming Zhang. 2023. Keep the Conversation Going: Fixing 162 out of 337 bugs for $0.42 each using ChatGPT. arXiv:2304.00385 [cs.SE]
[225] 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]
[226] 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]
[227] 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]
[228] Chenyuan Yang, Yinlin Deng, Runyu Lu, Jiayi Yao, Jiawei Liu, Reyhaneh Jabbarvand, and Lingming Zhang. 2023. White-box Compiler Fuzzing Empowered by Large Language Models. arXiv:2310.15991 [cs.SE]
[229] 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]
[230] 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
[231] 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]
[232] Yagmur Yigit, William J Buchanan, Madjid G Tehrani, and Leandros Maglaras. 2024. Review of Generative AI Methods in Cybersecurity. arXiv:2403.08701 [cs.CR]
[233] 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]
[234] 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
[235] 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]
[236] 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.
[237] Cen Zhang, Mingqiang Bai, Yaowen Zheng, Yeting Li, Xiaofei Xie, Yuekang Li, Wei Ma, Limin Sun, and Yang Liu. 2023. Understanding Large Language Model Based Fuzz Driver Generation. arXiv:2307.12469 [cs.CR]
[238] Chenyuan Zhang, Hao Liu, Jiutian Zeng, Kejing Yang, Yuhong Li, and Hui Li. 2023. Prompt-Enhanced Software Vulnerability Detection Using ChatGPT. arXiv:2308.12697 [cs.SE]
[239] He Zhang, Muhammad Ali Babar, and Paolo Tell. 2011. Identifying relevant studies in software engineering. *Information and Software Technology* 53, 6 (2011), 625–637.
[240] 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
[241] 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]
[242] 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]
[243] 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.
[244] 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]
[245] 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]
[246] 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).
[247] 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]
[248] 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]

[249] 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]

[250] 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]

[251] 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

[252] 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).